FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání diplomové práce

Název:	Bezpečnostní audit Ethereum projektu
Student:	Bc. Vojtěch David
Vedoucí:	Ing. Josef Gattermayer, Ph.D.
Studijní program:	Informatika
Obor / specializace:	Počítačová bezpečnost
Katedra:	Katedra informační bezpečnosti
Platnost zadání:	do konce letního semestru 2022/2023

Pokyny pro vypracování

Množství finančních projektů postavených čistě na technologii blockchain raketově roste. Problém je, že smart kontrakty jsou počítačový kód, který může obsahovat (a obsahuje) bugy. Kvůli decentralizované povaze není možné tento bug jednoduše opravit a jakmile je smart kontrakt spuštěn, je pod konstantním útokem hackerů. Vaším úkolem bude si takovýto audit zkusit na nějakém vzorovém smart kontraktu.

Pokyny:
- Seznamte se s technologií Ethereum a programovacím jazykem Solidity.
- Společně s vedoucím práce vyberte projekt přiměřeného rozsahu, proveďte jeho základní analýzu.
- Proveďte lokální deployment projektu (např. pomocí Truffle).
- Doplňte existující testy ke zvýšení test coverage >90% (například pomocí Brownie).
- Proveďte statickou analýzu kódu a kontrolu proti známým zranitelnostem (např. pomocí nástroje Slither).
- Napište fuzzy testy pro parametry funkcí i pořadí volání funkcí (např. pomocí nástroje Echidna).
- Proveďte manuální code review.

Elektronicky schválil/a prof. Ing. Róbert Lórencz, CSc. dne 13. listopadu 2021 v Praze.

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Diplomová práce

Bezpečnostní audit Ethereum projektu

Bc. Vojtěch David

Katedra informační bezpečnosti
Vedoucí práce: Ing. Josef Gattermayer, Ph.D.

27. dubna 2022

Poděkování

Děkuji celé svojí rodině, která mi poskytla zázemí, abych mohl tuto práci vypracovat a dobrou radou mi vždy pomohla. Dále svému vedoucímu za zajímavé zadáni a pomoc.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 2373 odst. 2 zákona č. 89/2012 Sb., občanský zákoník, ve znění pozdějších předpisů, tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo"), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu) licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 27. dubna 2022 . . . . . .

České vysoké učení technické v Praze
Fakulta informačních technologií

Odkaz na tuto práci

David, Vojtěch. *Bezpečnostní audit Ethereum projektu*. Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2022.

Abstrakt

Ve své práci se zabývám bezpečnostním auditem aplikace PWN, která je napsaná v jazyce Solidity a je nahraná na blockchain Etherea. V textu je detailně popsán princip protokolu Bitcoin a na těchto základech vysvětluji, jak funguje Ethereum a smart kontrakty, které jsou určeny pro nahrání na blockchain Etherea. Dále zkoumám fungovaní PWN aplikace, píšu pro ni unit testy a analyzuji výstup statické analýzy kódu z nástroje Slither. Následuje manuální code review, kde diskutuji slabiny návrhu aplikace, kritizuji odklonění implementace PWN od popisu ve white paperu a předkládám svůj návrh na vylepšení aplikace.

Klíčová slova Bitcoin, blockchain, POW, eliptické křivky, Ethereum, smart kontrakt, POS, DeFi, Solidity, Slither, PWN

Abstract

In this paper, I deal with security audit of PWN application, which is written in Solidity and deployed to Ethereum blockchain. The text describes in detail the principle of Bitcoin protocol. On that basis I explain Ethereum protocol and the smart contracts, which are meant to be deployed on the Ethereum blockchain. I analyze the PWN application, I write the unit tests for it and

I review the output of the static code analysis tool Slither. After that, I do a manual code review, where I discuss the weaknesses of the PWN design, I criticize that, the PWN implementation does not follow the description in the white paper and I propose my own application improvements.

Keywords Bitcoin, blockchain, POW, elliptic curve, Ethereum, smart contract, POS, DeFi, Solidity, Slither, PWN

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Když v roce 2008 anonym pod přezdívkou Satoshi Nakamoto přestavil ve svém white paperu koncept Bitcoinu, započal tím éru decentralizovaných měn a decentralizovaného finančnictví. Dokázal totiž vyřešit problém jak zařídit, aby se všichni uživatelé dokázali shodnout na společném stavu, takzvaně dosáhli konsenzu. V případě Bitcoinu je společný stav historie všech transakcí mezi uživateli. Bitcoin (a jeho příkladu následující kryptoměny) dosáhne této shody bez účasti centralizované autority a mohou se na ní podílet všichni uživatelé. Další problém, který efektivně vyřešil, je, že společný stav není reálně možné změnit škodlivým uživatelem.

Bitcoin je něco jako veřejná účetní kniha, kde jsou zaznamenány všechny transakce bitcoinu (měna používaná v protokolu Bitcoin) mezi uživateli a není možné tyto záznamy nijak měnit. Ethereum, o kterém je tato diplomová práce, používá stejný algoritmus pro dosažení konsenzu jako Bitcoin. Rozdílem je to, že tento stav neobsahuje pouze historii transakcí, ale i deterministické programy, které je možné pomocí transakce spustit a tyto programy (smart kontrakty) vykonají naprogramované operace. Ethereum touto vlastností otevřelo dveře decentralizovanému finančnictví, které může půjčovat prostředky, reprezentovat hodnotu v podobě tokenů a podobně. Smart kontrakty ale přinesly úskalí v podobně možných chyb a zranitelností v kódu, které mohou připravit uživatele o finanční prostředky. Například kvůli chybě ve smart kontraktu v aplikaci MonoX na decentralizované směňování tokenů, byly ukradeny tokeny v hodnotě 31 miliónů dolarů. Proto je potřeba smart kontrakty pečlivě auditovat, aby se předešlo podobným incidentům.

Motivací, proč jsem si vybral toto téma je, že se jedná o rychle se rozvíjející prostředí, kde se každým dnem objeví nový projekt nebo nápad. Bezpečnost a způsob, jakým jsou kryptoměny decentralizované, se opírá o kryptografické principy, a to je mi blízké vzhledem k mému zaměření studia.

V první kapitole vysvětlím, jak dosáhnout konsenzu decentralizovaným způsobem, který jsem nastínil v tomto úvodu, a jak je zaručena bezpečnost

celého protokolu. K tomu definuji potřebné koncepty jako jsou transakce, bloky, vytěžení bloku, blockchain, proof of work a jak se řeší rozvětvení blockchainu. V první kapitole je vše vysvětleno na Bitcoinu, ale koncepty se pro Ethereum neliší. V druhé kapitle definuji digitální podpis a představím eliptické křivky, které se v kryptoměnách využívají. Následující třetí kapitola spojuje v předchozích kapitolách vysvětlené pojmy a staví na nich další funkcionality, jako je spustitelný smart kontrakt a jazyk Solidity, ve kterém je možné smart kontrakty psát. Čtvrtá kapitola pojednává o možnostech vylepšení vlastností Etherea jako je možnost zpracovat více transakcí za sekundu a snížení poplatků. Také zde představuji náhradu způsobu dosažení konsenzu pomocí proof of work, kterou má být proof of stake, a uvedu rozdíly mezi těmito přístupy, výhody a nevýhody. Poslední kapitola je věnovaná aplikaci na decentralizované půjčky. Aplikace je implementovaná ve formě smart kontraktů a je napsaná v jazyce Solidity. Provedu bezpečnostní analýzu kódu, napíši testy, spustím automatickou statickou analýzu, vyjádřím se k návrhu aplikace a uvedu nápady pro zlepšení aplikace.

Cílem práce je pochopit protokol Ethereum a porozumět programovacímu jazyku Solidity. Dále vybranou aplikaci nasadit na testovací blockchain a otestovat její funkcionality. Dalším krokem je napsat unit testy s minimálním pokrytí kódu 90%, samotný kód podrobit automatické statické analýze a vyjádřit se k případným nalezeným zranitelnostem. Poté je cílem provést manuální code review a napsat fuzzy testy pro parametry funkcí a pro pořadí spuštění funkcí.

Kapitola 1

Bitcoin

Tato kapitola pojednává o konceptech, které stojí za Bitcoinem. Cílem této kapitoly je představit základní pojmy, které se vážou s kryptoměnami, abychom mohli na tyto základy navázat při popisu Etherea.

1.1 Co je to Bitcoin

Bitcoin je první úspěšná decentralizovaná digitální měna. Slovem decentralizovaná se myslí, že žádná důvěryhodná třetí strana není potřebná k tomu, aby mezi dvěma stranami proběhla transakce. Protože je odstraněna potřeba důvěryhodné třetí strana k provedení transakce, musí Bitcoin (a ostatní kryptoměny) přijít s protokolem, který zabrání dvojímu utracení, umožní dokázat vlastnictví bitcoinu[1] a další nezbytné vlastnosti potřebné pro použití jako digitální měna.

Měna bitcoin se používá v rámci Bitcoin protokolu pro uložení a směňování hodnot. Je pouze digitální, neexistuje ve fyzické podobě. Vzniká takzvaným procesem těžení, který popíšeme v podkapitole 1.6. Majitel bitcoinu má privátní klíče, kterými prokazuje vlastnictví a může jimi podepisovat další transakce, aby zaručil, že transakce nepopiratelně vnikla od něho.[5]

Bitcoin je distribuovaná peer-to-peer síť. Nenajdeme zde řídící servery, které by kontrolovaly správnost chodu sítě. Každý se může do Bitcoin sítě připojit, podílet se na jeho chodu a tím zvýšit decentralizovanost sítě. Bitcoin spojil dohromady koncepty z kryptografie a z distribuovaných systém, a výsledek jsou 4 hlavní pilíře, na nichž protokol stojí.[5]

- Decentralizovaná peer-to-peer síť
- Veřejné záznamy transakcí (blockchain)
- Pravidla pro nezávislou validaci transakcí a vydávaní nových bitcoinů.

[1] Aby se zamezilo zaměnění protokolu Bitcoin s měnou v něm používané, je zvykem protokol uvádět s velký "B" ale měnu s malým písmenem "b".

- Mechanizmus pro globální konsenzus pro shodu nad podobou blockchainu (proof of work)

Koncept Bitcoinu byl popsán v roce 2008 v publikaci *Bitcoin: A Peer-to-Peer Electronic Cash System*[6]. Publikace byla vydána anonymem, který používá pseudonym Satoshi Nakamoto. Na první pohled se může zdát, že protokol navržený anonymem je nedůvěryhodný. Ale protokol je postaven na prozkoumaných a ověřených konceptech a zdrojový kód je open source a každý k němu může přispívat. Neznalost identity Satoshi Nakamota může být považována spíše za výhodu, protože tento vynález nejde spojit s náboženstvím, rasou anebo státní příslušností. Identita Satoshi Nakamota tedy nemůže být zneužita v propagandě proti Bitcoinu.

První implementace protokolu byla vytvořena Satoshim v roce 2009[2] a byla zkontrolována mnoha vývojáři. V dubnu 2011 se Satoshi stáhl z veřejného prostoru a nechal vývoj už pouze na komunitě.[5]

1.2 Předchůdci Bitcoinu

Bitcoin není první projektem digitální měny, na následujících příkladech demonstrujeme, že Bitcoin se poučil z chyb předchozích nápadů, ale použil z nich ty dobré koncepty.

1.2.1 E-gold

Tento projet vznikl v hlavě onkologa Douglase Jacksona. E-gold je zajímavý tím, že používá fyzické drahé kovy jako krytí pro vystavené tokeny a tím zaručuje jejich hodnotu. Token Tether[3] je v tuto chvíli jedním z nejpoužívanějších tokenů se stabilním kurzem 1:1 vůči dolaru, čehož dosáhl tím, že za každý vydaný Tether kryje jedním dolarem. Princip je tedy podobný jako u E-goldu. Kromě lékaře byl Jackson také studentem ekonomické historie a byl přesvědčen, že zlato je nadřazeno papírovým penězům. Považoval za chybu, že státy se odklonily od zlatého standartu, neboť zlatu se nikdy nestalo, že by bylo považováno za bezcenné, jako se to stalo některým hyperinflacím postiženým měnám.[7]

Jacksonovým nápadem bylo vytvořit soukromou mezinárodní elektronickou měnu, která díky stabilitě ceny zlata bude odolávat volatilitě trhu. Uživatelé E-goldu si digitálně kupovali gramy zlata (později dalších cenných kovů), a ty mohly posílat dalším uživatelům E-goldu.[7] Všechny vklady byly skutečně kryty fyzickým cenným kovem, který byl uložen v bezpečnostních schránkách organizace.

[2] Zde je možno podívat se na první commit `https://github.com/bitcoin/bitcoin/tree/4405b78d6059e536c36974088a8ed4d9f0f29898`

[3] https://tether.to/en/

Nevýhodou bylo, že E-gold nebyl decentralizovaný, měl jedno řídící centrum, které úřady mohly odstavit a položit tím celý systém. Což se skutečně stalo, protože úřady chtěli E-gold regulovat a kontrolovat jeho uživatele. [7][8] Decentralizace tuto slabinu odstraňuje.

1.2.2 B-money

B-money je decentralizovaný finanční systém, který byl navržen v roce 1998 počítačovým inženýrem Wie Dai. Nikdy se B-money nedočkaly oficiálního spuštění. Ethereum má platidlo ether, jeden ether je 10^{18} wei, toto pojmenování je na počest zakladatele B-money.

Dai popsal B-money jako schéma, kde si skupina nevystopovatelných anonymů platí vzájemně penězi bez pomoci důvěryhodné třetí strany. [9]

Dai ve své práci popsal několik konceptů, které se v dnešní době staly standardem pro kryptoměny.

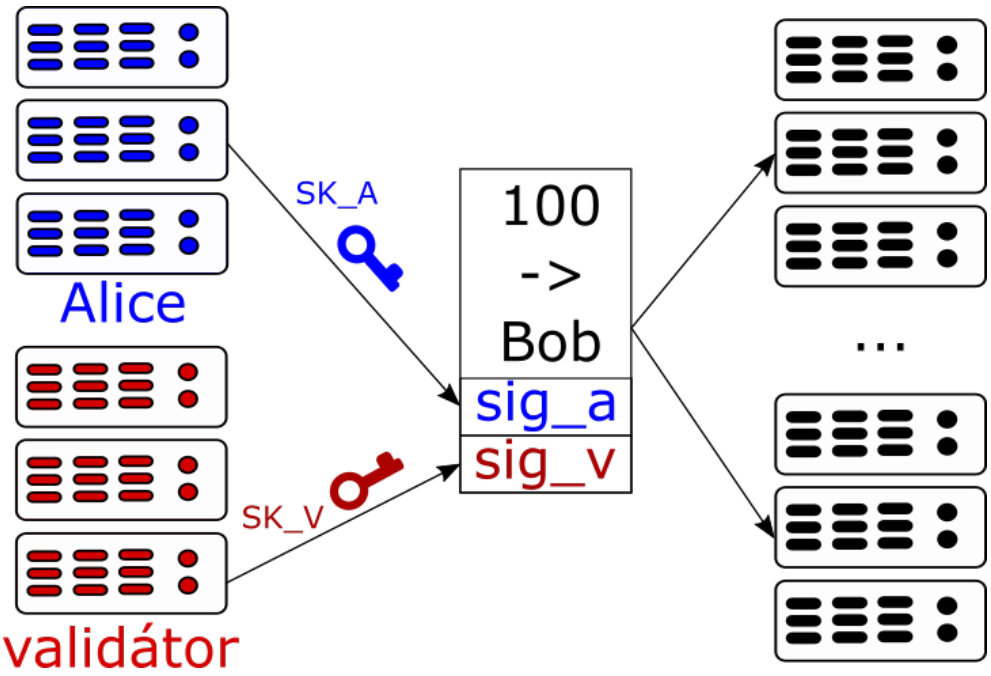
- Autentizace pomocí asymetrické kryptografie
- Veřejné záznamy transakcí, které společně validuje komunita
- Proof of work

Proof of work v tomto systému slouží pro vytvoření platidla, tím se liší od Bitcoinu, kde proof of work slouží k dosažení konsenzu, o této problematice mluvíme v podkapitole 1.6.2.3[10]

Transakce se podle Daie má provádět následovně.[10] Pokud Alice chce poslat X platidel Bobovi, tak všem uživatelům sítě (udělá broadcast) rozešle zprávu: "Posílám Bobovi X platidel". Poté tuto zprávu podepíše svým soukromým klíčem. Každý může díky veřejnému klíči ověřit, že tu zprávu skutečně vytvořila Alice a jedině Alice. Všichni uživatelé, kteří přijmou zprávu, validují podpis a že Alice má dostatečný zůstatek. Pokud je vše v pořádku, v lokální kopii databáze zůstatků všech účtů všichni uživatelé odečtou X platidel Alici a přičtou X platidel Bobovi. Tyto lokální databáze mají v sobě uloženou kopii globálního stavu celého protokolu a všichni účastníci musí mít tuto kopii stejnou, protože kdyby ne, tak by neseděly zůstatky jednotlivých účtů.[10]

Součástí transakce také musí být vybraný rozhodce. Podpis toho rozhodce je také součástí transakce. Jeho úkol je rozhodnout o validitě transakce, pokud nastane nějaký spor. Způsob, jakým se rozhodce vybírá, není v práci Daie popsán. Zřejmě to mělo být součásti pozdější diskuze. Pokud některá ze zmíněných stran (Alice, Bob, validátor) poruší svůj úkol, platí zbylým účastníkům reparaci. Částka této reparace je také stanovena v transakci. Provedení transakce je znázorněno na obrázku 1.1.[10]

Výpočetně náročný proof of work měl být založený na hashovacích funkcích, stejně jak je tomu v Bitcoinu, jak je detailně popsáno v podkapitole 1.6.2.3. Systém proof of work na získávání peněz se dělí na čtyři části.



Obrázek 1.1: Alice posílá peníze Bobovi

- Plánování. Všichni účastníci uvedou, kolik nových peněz by se mělo vytvořit. Pomocí makroekonomického modelu se udělá konsenzus, kolik nových peněz se vytvoří.
- Nabídka. Každý uživatel uvede, kolik peněz chce vytvořit společně s nevyřešeným problémem, který chce řešit. Každý problém by měl mít nějakou nominální hodnotu podle složitosti problému.
- Výpočet. Ti co vytvořili nabídku řeší definované úkoly.
- Vytvoření peněz. Uživatelé kontrolují řešení úloh, dokud nejsou vytvořeny všechny peníze, tak jak byl stanoveno v bodě plánování.

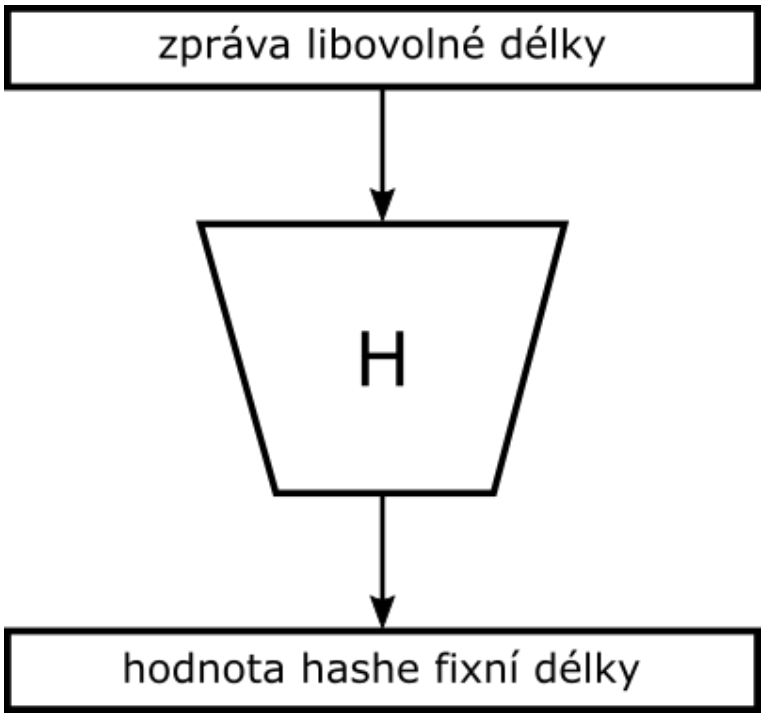
Jak je vidět z popisu, tak protokol není zcela dotažen do konce. Ale vyskytují se v něm hodně zajímavé koncepty decentralizace a proof of work, které se v pozměněné podobě vyskytují právě například v Bitcoinu.

1.2.3 Hashcash

Tento koncept, na rozdíl od E-goldu k nalezení v 1.2.1 a B-money popsaném v 1.2.2, není druhem elektronického platidla. Důvod proč hashcash v této práci uvádíme je, že v hashcash se používá algoritmu proof of work (POW), stejným způsobem jako v Bitcoinu, v čem se liší, je důvod k použití POW. Hashcash

využívá hashovacích funkcí k algoritmu proof of work (POW) a tím se bránit Denial of Service (DoS) útokům a posílání spamů velkému množství uživatelů. Bitcoinu používá POW jako součást algoritmu pro dosažení konsenzu. [11]

Definice 1 (Hashovací funkce) *Hashovací funkce je jednosměrné zobrazení, které zobrazí libovolně velký vstup na výstup s fixně danou velikostí (hash). Aby byla kryptograficky bezpečná, musí se vzor pro daný obraz hledat hrubou silou. Tedy zkoušet všechny možnosti.*



Obrázek 1.2: Hashovací funkce

Hashcash pracuje s tokenem, který prokáže, že byl stráven CPU čas při vytvoření tohoto tokenu. V případě posíláni emailu je myšlenka taková, že se do klasické hlavičky emailu přidá token a counter. Poté se počítá hash této hlavičky a inkrementuje se counter do té doby, než výsledná hash nemá k-bitů požadované hodnoty (tradičně nulové hodnoty), hodnota hashe se přiřadí tokenu. Jediným způsobem jak najít takovýto token je z definice 1 pouze hrubou silou. Protokol byl, navržen s hashovací funkcí SHA1 a aby bylo prvních 20 bitů tokenu nulových.[11]

Příjemce takového emailu jako první věc spočítá hash hlavičky emailu a poté ji porovná s tokenem. Pokud token a hash nesedí, je email zahozen. Pro odesilatele emailů je tedy ztíženo posílaní velkého množství emailů. Vygenerovaní tokenů stojí čas a peníze za elektřinu, ověření tokenu je však málo

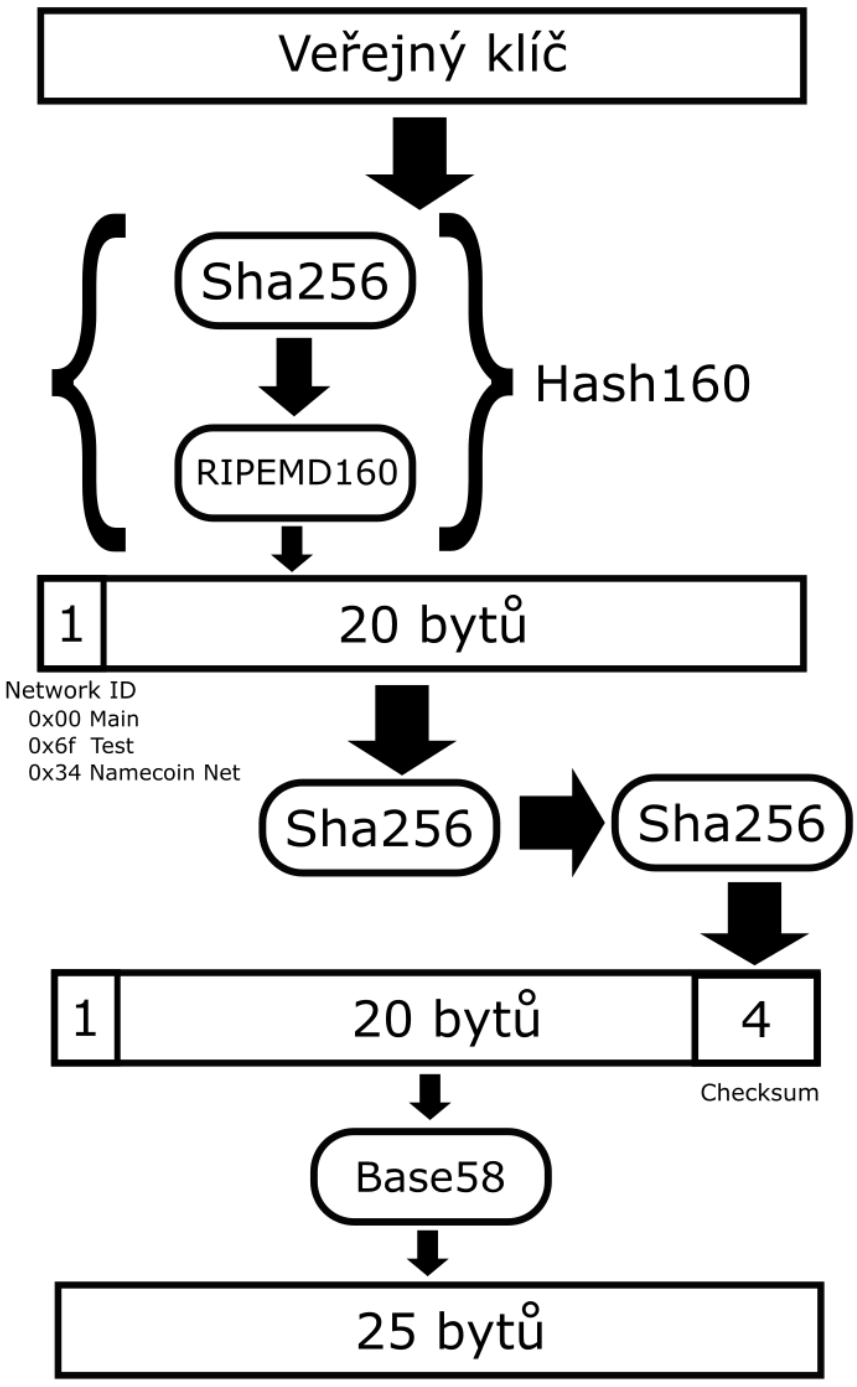
výpočetně náročné, v tom je veliká výhoda hashovacích funkcí. V případě, že je potřeba prodloužit čas generování tokenů v průměru n-krát, stačí požadovat o 2^{n-1} více nulových bitů.[11]CPU čas

1.3 Bitcoinová adresa

Bitcoinová adresa se používá jako unikátní identifikátor jednotlivých uživatelů. Reprezentuje uživatele v protokolu a není s ní spojen žádný osobní údaj, takže v tomto ohledu je anonymní. Záznamy transakcí ovšem anonymní nejsou, každý si může dopočítat, kolik bitcoinů daná adresa vlastní. Moderní peněženky pro zvýšení anonymity po každé transakci vygenerují novou adresu, které se používá pro další transakci. Když chce uživatel poslat bitcoin jinému uživateli, potřebuje znát jeho adresu, která je odvozena z veřejného klíče uživatele, který se zahashuje. V praxi je vysoce nepravděpodobné, aby adresa nebyla unikátní, musela by vzniknout kolize v kryptograficky bezpečné hashovací funkci. V době psaní diplomové práce byl jeden ze standardů odvození adresy z klíče zahashování veřejného klíče pomocí funkce *SHA256* a následně hashovací funkcí *RIPEMD-160*, spojením těch dvou funkcí se říká *HASH160*. [5] [12]

Dále se standardně spočítá checksum, který zabezpečí adresu proti případným překlepům při přepisování adresy nebo proti chybě při přenosu dat. Vezme se výstup ze *HASH160* a přidá se byte pro typ sítě a dvakrát se výsledná data zahashují funkcí *SHA256*. Z této hashe se vezmou čtyři byty a přidají se k adrese. V případě, že v zadání adresy je překlep, je to tímto detekována chyba, která je uživateli oznámena. Toto zabezpečení proti lidské chybě je velice praktické, protože všechny transakce jsou v Bitcoinu nevratné.[5]

Častým dalším krokem je zakódování výstupu z těchto dvou hashovacích funkcí do *Base58Check*. Tento krok se provádí pro čitelnější reprezentaci adresy. Base58 je odvozená od známé Base64. Base64 se používá pro převod binárních dat do textové podoby. Reprezentace Base64 obsahuje velké a malé znaky anglické abecedy, číslice a dva speciální znaky „+“ a „/“. Base58 je podmnožina Base64, ze které je odebraná množina znaků: 0, O, l, I, +, /. Jsou to znaky, které jsou vzhledově podobné znakům, které v množině zůstaly a mohou se zaměňovat. Odebráním se sníží pravděpodobnost chyby v případě, že adresu opisuje člověk.[5] Často se v peněženkách zobrazuje adresa zakódovaná do *Base58*, aby byla lépe čitelná a dalo se s ní textově pracovat. Celý postup pro odvození adresy z veřejného klíče je znázorněn na obrázku 1.3.[5]



Obrázek 1.3: Odvození Bitcoinové adresy z veřejného klíče

1.4 Bitcoinové transakce

Bitcoinová transakce stejně jako bankovní transakce slouží k přenesení určité částky bitcoinu od původního majitele k novému majiteli. Nový majitel může získaný bitcoin novou transakcí přenést k dalšímu majiteli. Transakce jsou nevratné, proto je potřeba dávat velký pozor na chyby při jejich vytváření.

Každá transakce obsahuje jeden nebo více vstupů a jeden nebo více výstupů. Výstup obsahuje částku bitcoinů, která se připíše novému uživateli a matematickou hádanku, kterou může vyřešit pouze majitel soukromého klíče, jemuž je transakce určena. Soukromý klíč odpovídá veřejnému klíči, ze kterého byla odvozena Bitcoinová adresa. Hádanka se také nazývá uzamčení transakce.[5]

Na vstupu najdeme hash transakce (její jednoznačný identifikátor) a řešení matematické hádanky, která dokazuje, že jsme oprávněni manipulovat s transakcí, kterou použijeme na vstupu.[6] Řešení se také říká odemčení transakce. [5] Je běžné, že součet bitcoinů vstupních transakcí je vyšší než součet bitcoinů výstupních transakcí. Rozdíl těchto hodnot je poplatek za transakci a připíše se uživateli, který validoval blok (vytěžil), ve kterém je transakce umístěna. Tento princip je popsán v podkapitolách blockchain 1.5 a těžení 1.6. [5]

Bitcoin na rozdíl od Etherea nefunguje tak, že si účastníci protokolu udržují zůstatky bitcoinů jednotlivých uživatelů. Místo toho si ukládají řetězce transakcí, ze kterých je možné zůstatky vždy dopočítat. Příklad řetězce transakcí je znázorněn na obrázku 1.5. Konci tohoto řetězce se říká unspent transaction output (*UTXO*). Jsou to výstupy z transakcí, které ještě nebyly použity jako vstup. Bitcoinová peněženka si udržuje přehled o *UTXO* u kterých může prokázat vlastnictví a zůstatek v peněžence je součet hodnot těchto *UTXO*.[5]

1.4.1 Vlastník bitcoinu

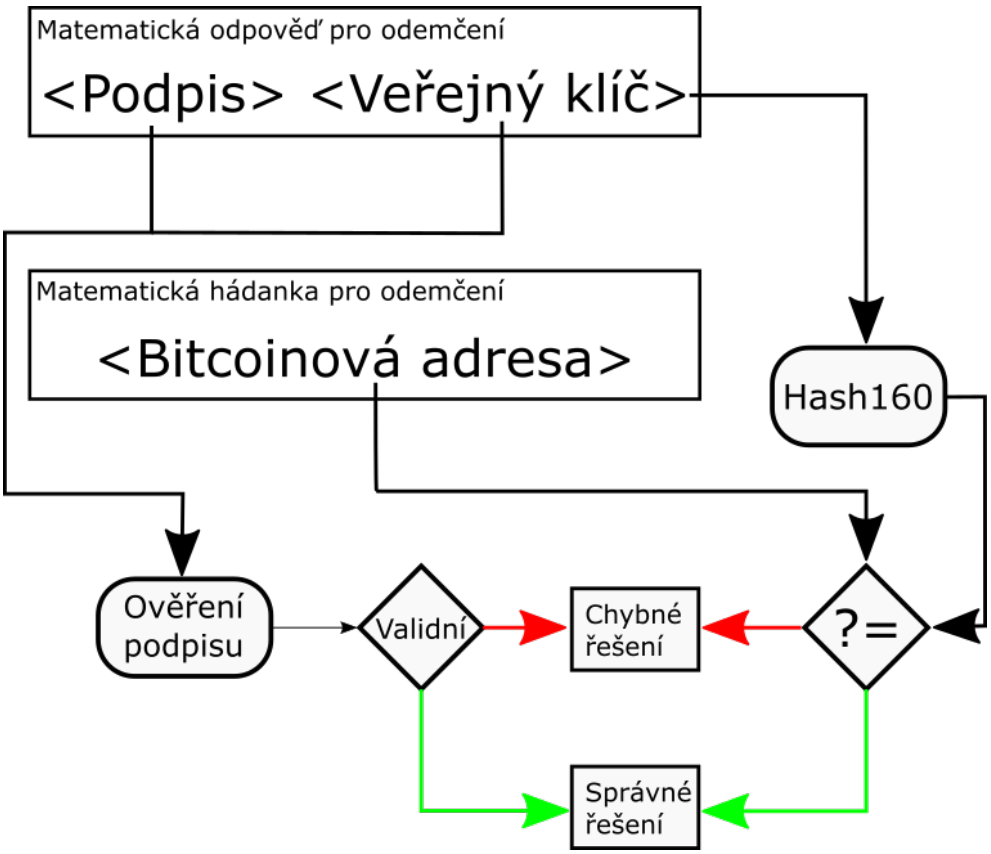
Jak je popsáno na začátku podkapitoly 1.4, každý výstup z transakce obsahuje matematickou hádanku, kterou může vyřešit pouze ten, komu je tato transakce určena. Tato hádanka se opírá o asymetrickou kryptografii. Té se budu věnovat více do hloubky v kapitole 2. Jak je popsáno v podkapitole 1.3, adresa vzniká z veřejného klíče. Veřejný klíč je spočítán ze soukromého klíče a pouze se znalostí veřejného klíče není možné spočítat klíč soukromý. Hádanka na výstupu je pouze Bitcoinová adresa toho, komu je transakce určena. Řešení je důkaz, který ukáže, že já jsem majitel soukromého klíče ze kterého byl dopočítán klíč veřejný a z něho dopočítána Bitcoinová adresa. Toto řešení obsahuje podpis transakce, kterou uživatel používá na vstupu a veřejný klíč. Postup pro ověření, že odemčení transakce bylo provedeno správným soukromým klíčem, je následující:

- Ověří se, že je možné z veřejného klíče odvodit Bitcoinovou adresu. Postup pro odvození adresy je popsán v 1.3.

- Pomocí veřejného klíče se ověří korektnost podpisu. Přesný postup jak ověřit podpis transakce je popsán v podkapitole 2.1.1.

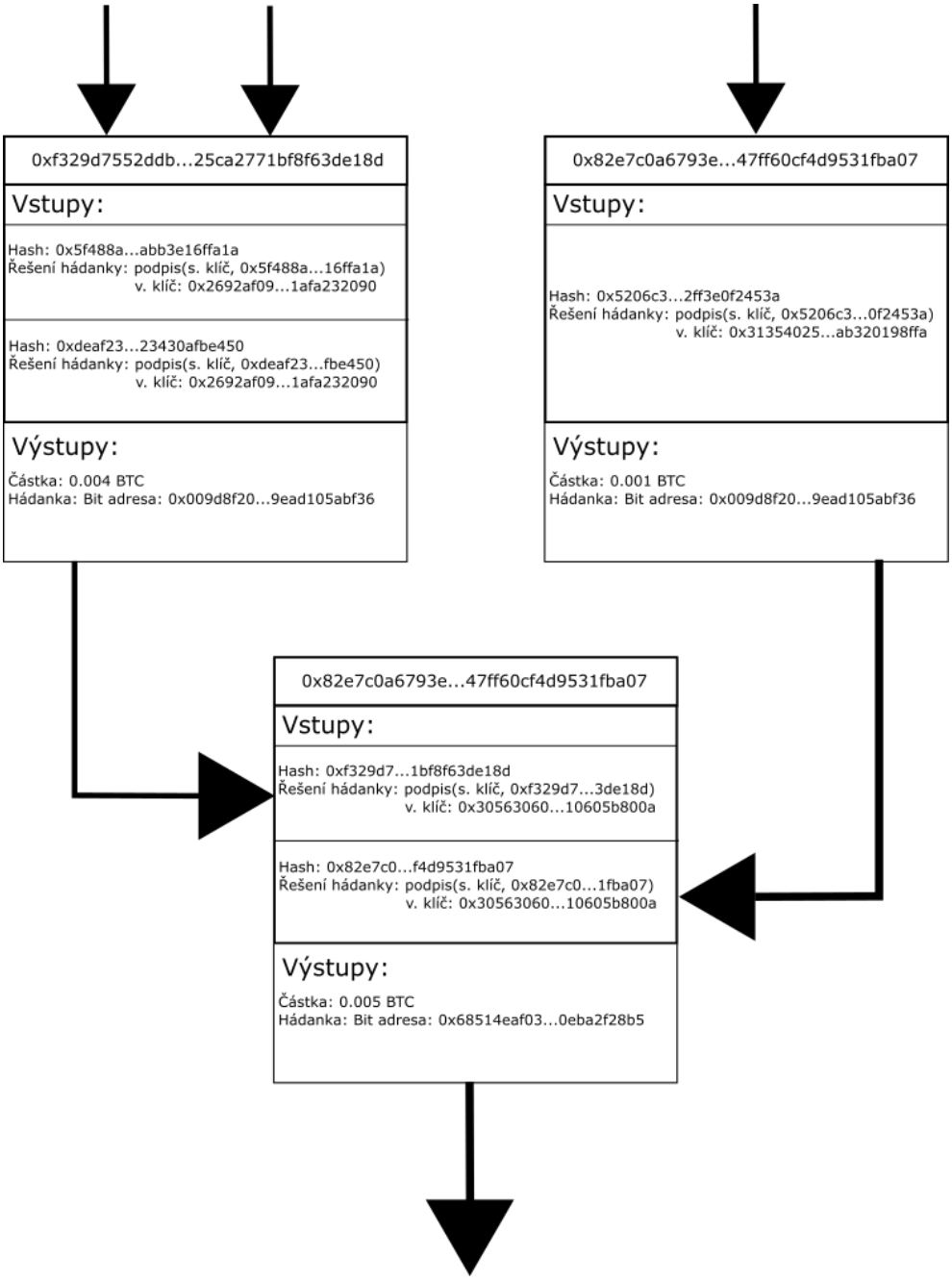
- Pokud oba předchozí body jsou v pořádku, je řešení hádanky validní.

Pouze majitel správného soukromého klíče může spočítat veřejný klíč, který odpovídá Bitcoinové adrese a zároveň je možné pomocí tohoto samého veřejného klíče ověřit podpis transakce. Celý postup pro odemčení transakce je znázorněn na obrázku 1.4.



Obrázek 1.4: Ověření řešení hádanky

Prozrazení soukromého klíče vede k tomu, že každý, kdo tento soukromý klíč zná, může posílat *UTXO* kam se mu zlíbí, protože je schopen odemknout dané transakce. Proto je velice důležité žádný soukromý klíč nikdy nikomu neprozrazovat.



Obrázek 1.5: Trasnsakční řetěz

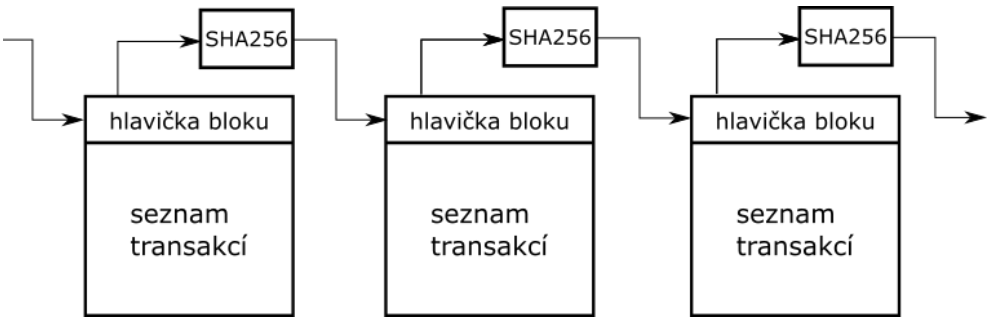
1.4.2 Rozměnění transakce

Velice dobrá analogie pro *UTXO* jsou bankovky v běžném peněžním světě. Když chci penězi zaplatit částku 600 a mám jen například bankovky v hodnotě 500 a 200, musí mi prodejce 100 vrátit. Nelze prodejci dát jen půlku bankovky v hodnotě 200 a tvrdit, že nyní to má hodnotu 100.

To samé platí i pro *UTXO*. Všechny volné transakce, se kterými mohu manipulovat, mají svoji pevně danou hodnotu, která je definovaná na výstupu z transakce. Pokud chci platit nějakou částku, musím z těchto volných transakcí tuto částku přesně poskládat. Pokud to není možné, na vstup dáme transakce, které placenou částku převyšují. Nějakou sumu ponecháme pro poplatek (o poplatcích budeme mluvit později v podkapitole 1.5) a pro zbylou částku vytvoříme výstupní transakci směřovanou na Bitcoinovou adresu, kterou si sami určíme, zpravidla na tu samou odkud transakci posíláme. Obrázek 1.5 ukazuje, jak se výstupy z transakcí používají jako vstupy do dalších transakcí.

1.5 Blockchain

Blockchain je databáze, která v sobě ukládá všechny transakce, které kdy v historii Bitcoinu proběhly. Protože jsou tato data uložena, mohou uživatelé kontrolovat, jestli jsou na vstupu transakcí pouze *UTXO* a tím nedošlo ke dvojímu utracení bitcoinu. Blockchain se skládá z bloků, které mají v sobě uložené záznamy o jednotlivých transakcích a metadata neboli hlavičku bloku. Nová data se do blockchainové databáze dají přidávat pouze po blocích. Součástí hlavičky bloku je hash hlavičky předchozího bloku, nonce, hodnota současné časové známky a kořen Merkleova stromu. Nonce a Merkleův strom bude později vysvětlen v 1.5.1 a 1.6. Tím, že blok v sobě uchovává hash hlavičky předchozího bloku, se z bloků stává seřazený zpětně orientovaný spojový seznam. Můžeme tedy dohledat všechny předchozí bloky podle hlaviček bloků a projít tímto způsobem celou historii Bitcoinu. [6]

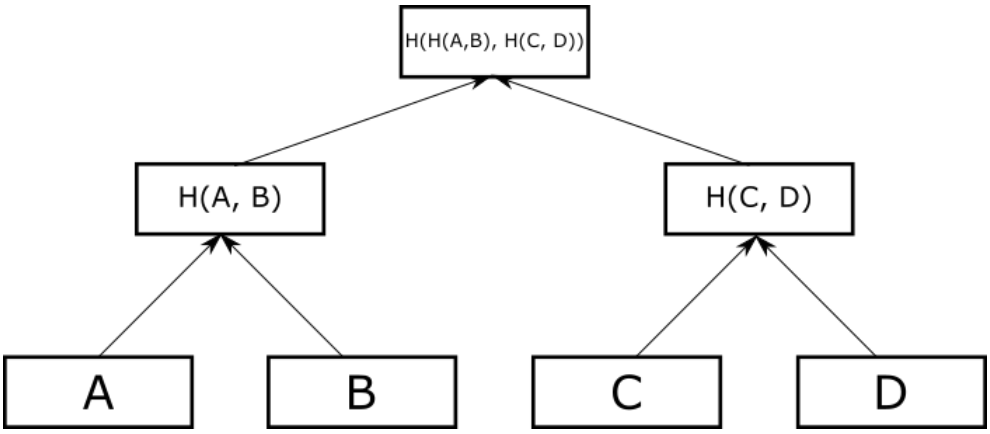


Obrázek 1.6: Blockchain

Bloky se identifikují podle hashe jejich hlavičky a nebo podle pozice v blockchainu. Nultou pozici má úplně u první blok (Genesis blok) a pak podle vzdálenosti od tohoto prvního bloku je možné bloky identifikovat. Vzdálenosti bloku od Genesis bloku se říká výška bloku.

1.5.1 Merkleův strom

Merkleův neboli hashový strom je datová struktura, která se používá pro sumarizaci transakcí. [6] Jedná se o binární strom, který se vytváří rekursivně od listů ke kořenu stromu. Začíná se s daty, v našem případě jsou to transakce, které považujeme za listy stromu. Listy se po dvojicích zahashují a výsledná hash je rodičovský vrchol této dvojice. Rodičovské vrcholy opět po dvojicích zahashujeme a výsledek je jejich rodičovský vrchol. Takto se pokračuje až zbyde jen jeden vrchol a to je kořen Merkleova stromu.[13] Ilustrováno na obrázku 1.7.

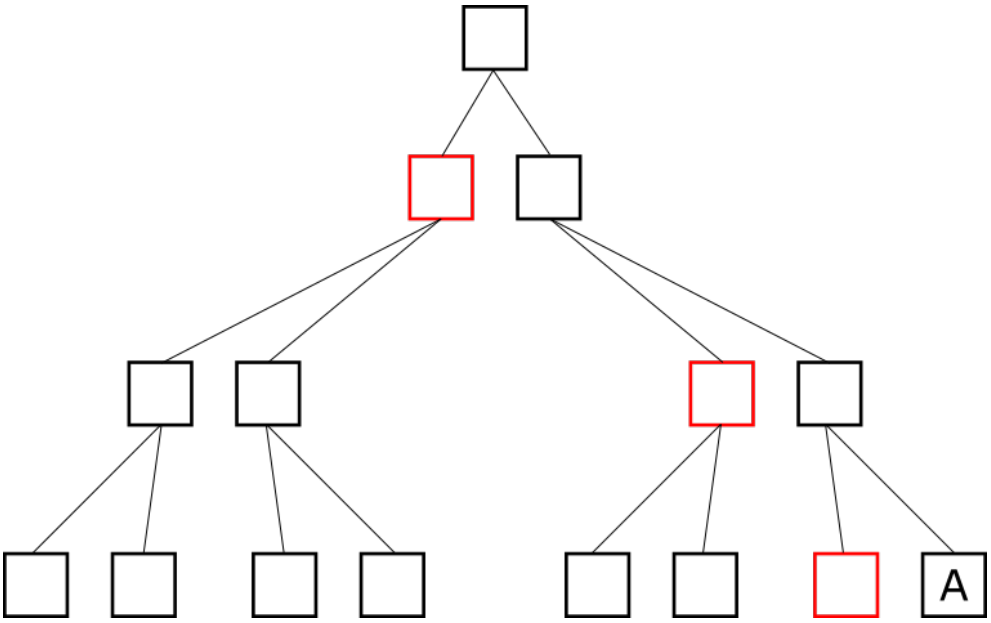


Obrázek 1.7: Merkleův strom

Pro důkaz, že některá specifická transakce je součástí konkretního bloku, není potřeba mít celý blok se seznamem transakcí. Stačí mít Merkleovu cestu a hodnotu kořene Merkleova stromu. Tato cesta je seznam všech vrcholů potřebných pro spočítání hodnoty kořene Merkleova stromu. Počet těchto vrcholů v této cestě je roven $log_2(n)$, kde n je počet všech vrcholů v Merkleově stromu.[5][6][13]

Tento způsob ukládání otisků všech transakcí v hlavičce bloku využívá v Simplified Payment Verification (SPV). Je nepraktické, aby každý node (uživatel, který provozuje Bitcoinový protokol) měl u sebe uložený blockchain v celém rozsahu. Node který používá SPV a nemá celou kopii blockchainu, se nazývá odlehčený node a naopak node s uloženým blockchainem je plný node. Odlehčený node si pouze ukládá hlavičky všech bloků, ale je i tak schopný zkontrolovat, jestli se transakce v bloku nachází. Verifikace se provede tak, že

požádá plný node o Merkleovu cestu k dané transakci v konkretním bloku. Nyní je možné spočítat kořen Merkleova stromu a porovnat jej s hodnotou kořene, který se nachází v hlavičce bloku, kterou má odlehčený node uloženou. Na obrázku 1.8 je znázorněno, které vrcholy musí být v Merkleově cestě pro dopočítání kořene.[5]



Obrázek 1.8: Merkleova cesta

Tento odlehčený node využívají běžně například peněženky, které jsou často na mobilních telefonech, takže nemají dostatek paměti na uložení celého blockchainu. Odlehčený node také nemůže verifikovat celou historii transakcí, a tak nemůže zjišťovat, jestli nebyla nějaká transakce utracena vícekrát a nemůže se účastnit procesu těžení a nemůže tak získat odměnu za vytěžený blok. Může ale udržovat zůstatek bitcoinů jednotlivých Bitcoinových adres, protože se může doptávat plných nodů na to, jestli je transakce součástí daného bloku a tedy jestli je v historii Bitcoinu.

Pokud by někdo chtěl změnit obsah transakcí, změnil by se tím i kořen Merkleova stromu, tím by se změnila i hash hlavičky bloku. To by změnilo hash hlavičky následujícího bloku a tak dále. Tímto by se musel pozměnit nejenom blok, ve kterém se mění transakce, ale i všechny následující bloky. Pro pozměnění bloku je potřeba vykonat POW, tento pojem souvisí s těžením, který je popsána v následující podkapitole 1.6.

1.6 Těžení

Do této chvíle popsaný systém transakcí není dostačující, protože nedokáže zamezit tomu, aby vlastník *UTXO* neutratil tuto *UTXO* vícekrát. Uživatel může vytvářet stále dokola transakce, kde je vstupem jedna a ta samá transakce a k ní uvést správné řešení pro odemčení transakce. Nezavedli jsme zatím žádný proces, který by toto kontroloval. Způsob, jakým by se tomuto dalo zamezit, je zavést centrální autoritu, která by kontrolovala všechny transakce a validovala by je. Toto je ovšem proti decentralizované myšlence Bitcoinu.[6]

Je tedy potřeba zavést jednu společnou historii a pořadí všech transakcí mezi všemi účastníky protokolu. [6] Všechny nově vytvořené transakce se zveřejňují. Je však nutné, aby tyto transakce někdo validoval a také aby se dokázalo dojít ke společné shodě, co jsou validní transakce. Tyto validátory je nutné motivovat odměnou. Tomuto procesu se říká mining neboli těžení a uživatelé kteří transakce kontrolují a provozují již zmiňovaný plný node se nazývají mineři nebo těžaři.

1.6.1 Coinbase

Těžaři kontrolují, zda vstupy do nové transakce obsahují správné řešení pro odemčení transakce a z databáze všech transakcí ověří, jestli již vstupy nebyly někdy utraceny. Nové transakce těžaři poté skládají do bloku a tento blok se přidá na konec blockchainu. Za přidání do blockchainu a validování všech transakcí si těžař připíše odměnu.

Odměna se skládá ze součtu poplatků ze všech transakcí v daném bloku a ze speciální transakce zvané coinbase. Coinbaise je první transakce v bloku, která je připsána minerovi, který daný blok vytěžil. Nemá žádné vstupy a na výstupu má předem definovanou částku bitcoinu a adresu těžaře. Bitcoin z coinbasu není nikdy předtím utracený a tímto způsobem se bitcoin vytváří.[5]

Počet bitcoinů se v coinbasu jednou za 210 000 bloků sníží na polovinu. Na začátku Bitcoinu byla odměna 50 bitcoinů za vytěžený blok. V roce 2022 je tato částka 6.25 bitcoinů.[14]

1.6.2 Konsenzus

V podkapitole 1.5 je popsán systém vytváření bloků a jejich ukládání do blockchainu. Jak ale poznat, který blok je validní a jaké je jejich správné pořadí, když není centrální autorita která o tomto rozhoduje? Je potřeba dosáhnout společné shody mezi nody o tom, který následující blok je správný. Této shodě se říká konsenzus. Všechny nody nezávisle na sobě provádějí následující seznam úkonů, který vede k dosažení konsenzu.[5]

- Ověření každé nové transakce.
- Agregování těchto nových transakcí do bloku.

- Ověřování validity všech nových bloků.
- Vybrání toho řetězce bloků kde je vykonáno nejvíce práce.

V následujících podkapitolách projdeme jednotlivé položky daného seznamu a do většího detailu je vysvětlíme.

1.6.2.1 Ověření nové transakce

Když vznikne nová transakce, tak je odeslána (broadcast) všem plným nodům. Když node novou transakci obdrží, tak jako první zkontroluje, že vstupy jsou skutečně pouze *UTXO*, součet vstupů je větší nebo roven výstupu, řešení pro odemčení transakce platí a další úkony týkající se spíše technického rázu protokolu než konceptu Bitcoinu. Poté transakci node pošle sousedním nodům, se kterými je propojen.[5]

1.6.2.2 Agregování transakcí do bloku

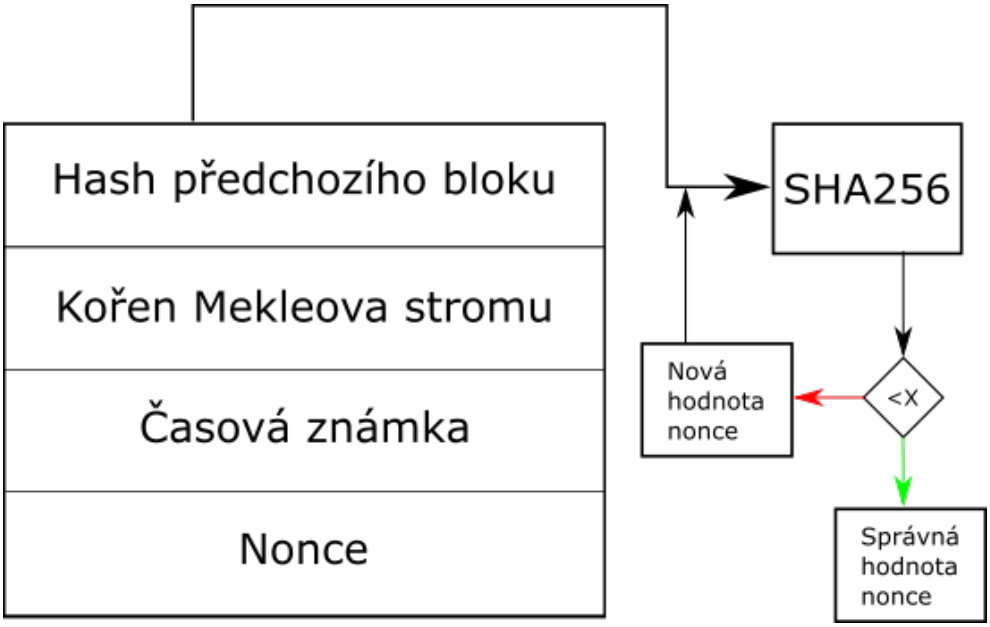
V momentě, kdy byl nový blok přidán do blockchainu, všechny nody vytvoří nový kandidátní blok z transakcích, které ještě nebyly vloženy do bloku. Každý plný node si vytvoří svůj vlastní. Pořadí a které transakce vybrat je na daném nodu. Pravidlem bývá, že větší prioritu pro zaražení do bloku mají transakce s vyšším poplatkem. Dále se vytvoří první transakce neboli coinbase a na závěr se vznikne hlavička bloku, která mimo jiné obsahuje hash bloku, na který bude tento blok navazovat. [5].

1.6.2.3 Proof of work

Po vytvoření kandidátního bloku se musí spočítat důkaz pro algoritmus POW, který dokáže, že pro vytvoření kandidátního bloku byla vynaložena výpočetní práce. Protože je požadováno odvedení výpočetní práce pro vytvoření bloku, není možné vytvořit mnoho falešných uživatelů, kteří budou potvrzovat a vytvářet podvržené bloky, jak by tomu mohlo být v případě pouhého hlasování všech uživatelů. Takto musí nepoctivý uživatel mít větší výpočetní sílu něž zbytek sítě Bitcoinu. Jakým je to reálným nebezpečím je více popsáno v podkapitole 1.7.

Pro POW je opět použita hashovací funkce, protože vypočítání důkazu je náročné, ale ověření není. Hashuje se hlavička kandidátního bloku stále dokola, přičemž se v ní mění hodnota nonce. Cílem je získat hash hlavičky, která bude odpovídat předem zadanému formátu. V Bitcoinu se požaduje, aby hodnota hashe byla menší než cílová hodnota. Čím menší požadovaná hodnota, tím je potřeba více nulových prvních bitů hashe. Pokud se nodu podaří najít hodnotu nonce splňující tuto podmínku, tak je blok vytěžený a je odeslán všem ostatním nodům. Nody zkontrolují, jestli je POW správný (tento proces ověření je již výpočetně nenáročný, protože stačí jednou spočítat hash

hlavičky kandidátního bloku) a zda jsou transakce validní podle postupu v 1.6.2.1. Pokud kontrola projde, nody přestanou pracovat na svém kandidátním bloku, tento blok vloží na konec blockchainu, vytvoří nové kandidátní bloky s výškou o jedna vyšší a těžení začíná nanovo. [5]



Obrázek 1.9: Proof of work

Hodnota proti které se hledá POW se může měnit podle rychlosti těžení. Cílem Bitcoinu je, aby se blok vytěžil přibližně jednou za 10 minut. Pokud se bloky těží moc rychle, je hodnota snížena o jeden řád ve dvojkové soustavě, tím se v průměru dvakrát zvýší potřebná práce pro vytěžení bloku.[5]

1.6.2.4 Rozvětvení blockchainu

Vzhledem k tomu, že Bitcoin je decentralizovaný, je možné, že různé nody mají různé bloky uložené v blockchainu. To může například nastat tak, že nejrychleji vytěžený blok přijde nodu se zpožděním a v mezičase node přijal jiný validní blok jako nový vrchol blockchainu. Poté nastává větvení blockchainu, protože není mezi všemi nody konsenzus, který blok je vrcholem blockchainu.

Node vždy vytváří kandidátní blok pro tu větev, kde je nejvíce odvedené práce. Tedy na té větvi, která je nejdelší.[5][6]

Například pokud node A vytěží validní blok A_1 v podobném čase jako node B vytěžil validní blok B_1, tak nody kterým přišel první blok A_1 pracují na bloku A_2, který má jako předka blok A_1. Až jim později přijde blok B_1, ověří že je validní, uloží si ho, kdyby náhodou větev s B_1 byla ta delší, ale

dále pracují na A_2. Nody, které dostaly jako první blok B_1, pracují na novém bloku B_2.

V tuto chvíli začíná závod mezi nody kteří hledají A_2 a B_2, jedna část nodů pracuje na větvi A a druhá na větvi B. Ta větev, kde se vytěží další validní blok, se stane delší větví a kandidátní bloky se vždy tvoří na větvi, kde je více vytěžených bloků. Větev, která v našem příkladu vyhrála, je považovaná za hlavní větev blockchainu a tvoří se kandidátní bloky již pouze pro ni.[5]

Společně s POW toto tvoří algoritmus pro konsenzus nad stavem blockchainu. Tomuto konsenzu se říká Nakamotův konsenzus. Pro jednoduchost se často algoritmu pro nalezení konsenzu říká pouze POW. Čím více uživatelů se připojí k Bitcoinu, tím je více decentralizovaný, vykonává se více POW a je i tím více bezpečný. Teto koncept decentralizovaného konsenzu se v praxi ukazuje jako funkční a osvojily si ho další kryptoměny jako například Ethereum.

1.7 51% útok

Bitcoin předpokládá, že většina výpočetní síly tvoří a validuje bloky poctivě. Pokud by ale měl útočník většinu výpočetní síly, je možné vytvořit větvení a změnit historii blockchainu. Tento útok neumožňuje vytvářet falešné transakce, nody by tyto bloky nikdy nepřijaly jako validní, protože by nesedělo odemčení transakce.[5]

Je teoreticky možné upravovat vlastní transakce. Například útočník si za bitcoin pořídil nějaký produkt. Počká, až produkt dostane a poté vytvoří větvení hned před blokem, kde byla jeho transakce zaznamenána a vytěžena. Vytvoří alternativní blok, kde jeho transakce je s nižší částkou, nebo není vůbec. Nyní však musí vytvářet a těžit bloky rychleji než zbytek sítě, aby dosáhl toho, že jeho alternativní větev je delší než hlavní a tím se stane hlavní větví. Bitcoin utracený za produkt tak může utratit znovu.

Proveditelnost tohoto útoku je závislá na velikosti Bitcoinové sítě. Čím více uživatelů se podílí, tím těžší POW se řeší a tím je finančně náročnější útok provést. Čím větší decentralizace tím větší bezpečnost protokolu.

Za rok 2022 je předpokládáno, že se těžením spotřebuje 204.50 TWh [4] energie. To je roční energetická spotřeba Thajska. Pokud někdo chtěl provést 51 % útok, potřeboval by více než polovinu této energii. To je v tuto chvíli na útok ze strany jednotlivců nebo i skupin nereálné.

Na druhou stranu kvůli velice malé pravděpodobnosti, že by jednotlivec mohl závody v POW vyhrát, vznikají miningové pooly neboli sdružení minerů. Pokud by se spojily 4 největší pooly[5], měly by více než 50 procent výpočetní síly. Decentralizace Bitcoinu se paradoxně kvůli příliš velké náročnosti POW zmenšila.

[4] `https://digiconomist.net/bitcoin-energy-consumption/`
[5] `https://btc.com/stats/pool`

Ekonomický pohled je ale optimističtější. Mineři dostávají odměny v bitcoinech, není tedy v jejich zájmu pošramotit pověst Bitcoinu a tím snížit jeho cenu na trhu.

Kapitola 2

Eliptické křivky

V kapitole 1 je popsán protokol Bitcoin. Jeho bezpečnost se silně opírá o asymetrickou kryptologii. V současné implementaci se používají eliptické křivky pro veřejné a soukromé klíče [15]. V této kapitole je popsáno, jak asymetrická kryptografie na eliptických křivkách funguje.

2.1 Asymetrická kryptografie

Základním kamenem asymetrické kryptografie je pár veřejného a soukromého klíče. Veřejný klíč je rozdistribuován mezi ostatní strany a soukromý klíč je držen v tajnosti. Pro šifrovaní se používá klíč veřejný a pouze majitel klíče soukromého dokáže takovou zprávu dešifrovat. V tomto je největší rozdíl od symetrického šifrovaní, které šifruje i dešifruje vždy stejným klíčem. [16]

Musí být výpočetně neschůdné bez znalosti soukromého klíče šifrový text dešifrovat nebo odvodit soukromý klíč. Po splněnou obou podmínek můžeme asymetrickou šifru považovat za bezpečnou.

2.1.1 Digitální podpis

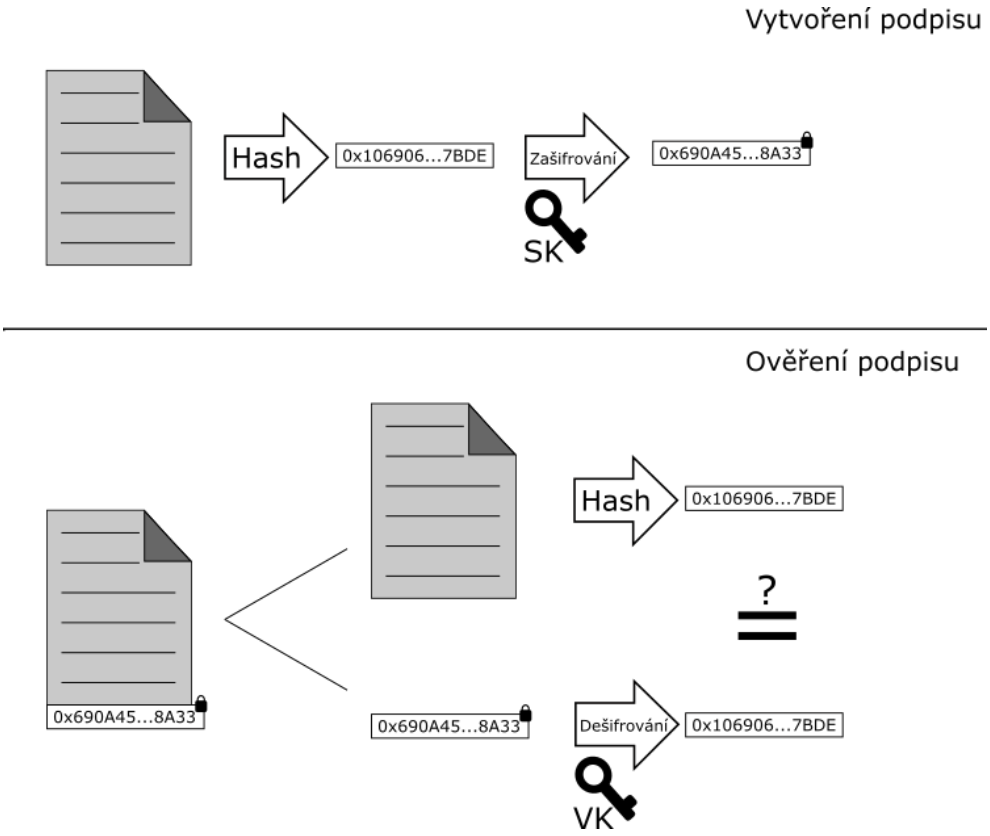
Významnou aplikací asymetrické kryptografie je digitální podepisování. Podepisuje se klíčem soukromým a podpis se ověřuje klíčem veřejným. Digitální podpis musí mít následující vlastnosti.[17]

- Autentizace. Nikdo jiný než majitel soukromého klíče nedokáže daný podpis vytvořit.
- Integrita. Obsah podepsané zprávy se nedá změnit aniž by byl podpis stále validní.
- Nepopiratelnost. Autor podpisu nemůže popřít, že by podpis vytvořil.

Používaným algoritmem pro vytvoření digitálního podpise je spočítat hash zprávy a tu zašifrovat soukromým klíčem. Výsledná sekvence bytů se připojí

za zprávu. Po verifikaci podpisu se opět spočítá hash zprávy a podpis se dešifruje veřejným klíčem.[17] Tento postup je zobrazen na obrázku 2.1. Hash a dešifrovaný podpis musí mít stejnou hodnotu. Hashovací funkce jsou definované v kapitole 1 definice 1.

Hashovací funkce zaručí integritu, zpráva nejde měnit aniž by se nezměnila hash zprávy. Šifrování soukromým klíčem zajistí autentizaci a nepopiratelnost. Nikdo jiný než majitel soukromého klíče by nedokázal podpis vytvořit.[17]



Obrázek 2.1: Vytvoření a ověření digitálního podpisu

2.2 Eliptické křivky nad reálnými čísly

Nyní se již zaměříme na konkrétní příklad asymetrické kryptografie a to jsou eliptické křivky (ECC). ECC se standartě v kryptoměnách používají, protože klíče mají kratší bitovou délku než například jiná asymetrická šifra RSA a tak se ušetří místo v blockchainu. Dále jsou rychlejší než klasické asymetrické systémy (RSA). Prvně si definujeme co znamená eliptická křivka nad $\mathbb{R}$.

Definice 2 (Eliptická křivka) *Eliptická křivka nad tělesem $\mathbb{R}$ je množina všech bodů $E \subset R^2$ splňující rovnici:*

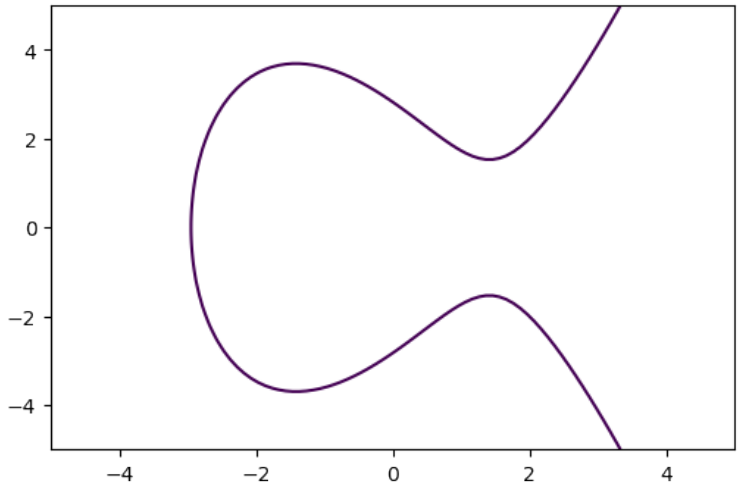
$$y^2 = x^3 + ax + b, \quad (x, y) \in \mathbb{R}^2$$

kde $a, b \in \mathbb{R}$ splňují $4a^2 + 27b^2 \neq 0$

Příklad takovéto křivky je zobrazen na obrázku 2.2. Tvar rovnice uvedené v definici 2 se nazývá Weierstrassovy rovnice. Obecnější typ eliptické křivky $y^2 = x^3 + ax^2 + bx + c$ se také používá, ale je možné ho jednoduchou algebraickou úpravou na čtverec a poté po transformaci souřadnic převést do Weistrassova tvaru s jinými koeficienty. [18][19][5]

$$x^3 + ax^2 + bx + c = \left(x + \frac{a}{3}\right)^3 + \left(b - \frac{a^2}{3}\right)x + c - \frac{a^3}{27}$$

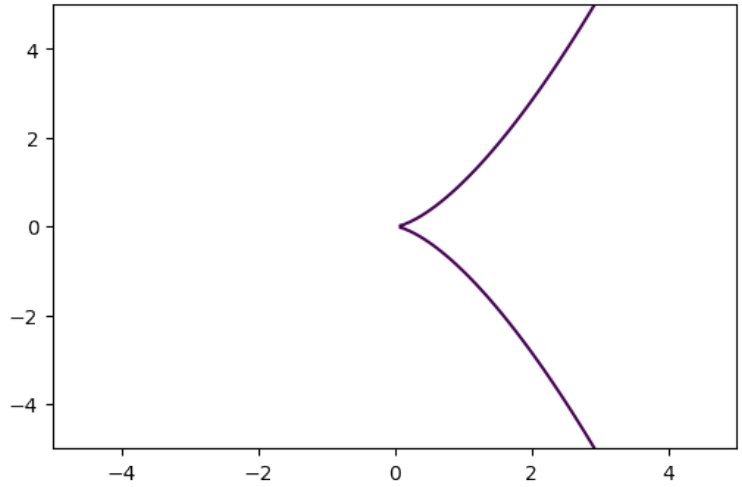
$$(x, y) \mapsto \left(x - \frac{a}{3}, y\right)$$



Obrázek 2.2: $y^2 = x^3 - 5x^2 + 8$

Podmínka $4a^2 + 27b^2 \neq 0$ napsaná v definici 2, říká, že kubický diskriminant $4a^2 + 27b^2$ musí být různý od nuly. Kubický polynom má tři navzájem

různé komplexní kořeny právě tehdy, když diskriminant není roven nule. Tato podmínka zaručí, že křivka nemá průnik sama se sebou, takzvaný singulární bod. Vznikne tak nesingulární (hladká) křivka.[19][18] To je důležité pro sčítání bodů na křivce, které definujeme v následující podkapitole. Příklad křivky se singulárním bodem je zobrazen na obrázku 2.3.



Obrázek 2.3: $y^2 = x^3$

2.3 Sčítáni dvou bodů na křivce

Pokud na body křivky E zadefinujeme sčítání $\oplus : E \times E \mapsto E$ a přidáme do množiny speciální bod $\mathcal{O}$, poté tato množina tvoří abelovskou grupu. Pro geometrickou představu lze sčítání bodu P a Q popsat následovně.[18]

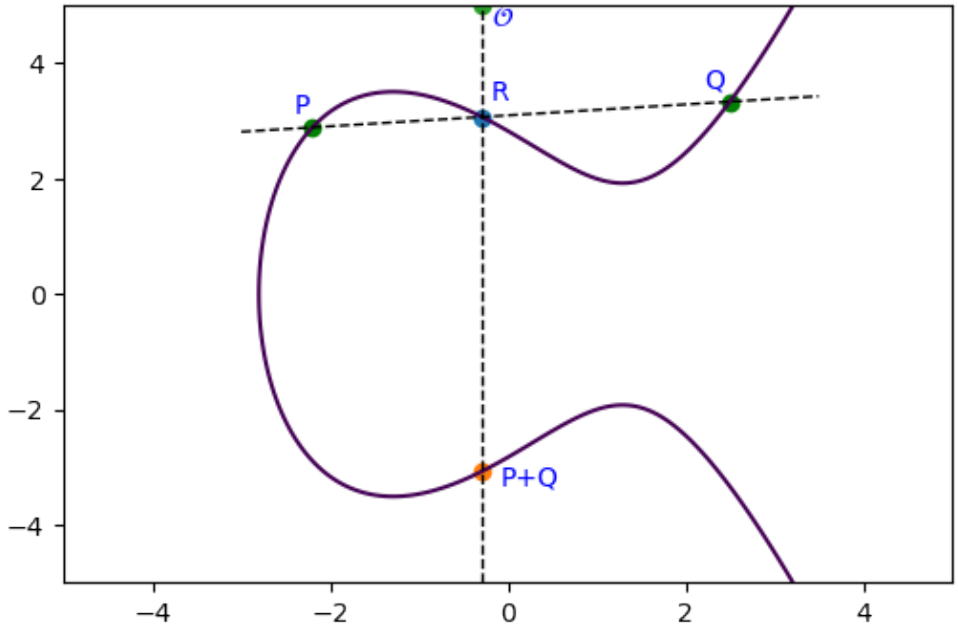
- Vytvoříme přímku procházející body P a Q.
- Vezmeme průsečík s E jiný než jsou body P a Q a pojmenujeme ho bod R.
- Bod R zrcadlíme podle osy x na křivku E a tím vznikne bod $P \oplus Q$.

Pokud body P a Q mají stejnou souřadnici x ale jiné y, poté neexistuje jiný průsečík na křivce E než je body P a Q. V tuto chvíli definujeme, že výsledek součtu je speciální bod $\mathcal{O}$. Označuje se také jako bod v nekonečnu. [18]

Součet bodů P a Q je zobrazen na obrázku 2.4. Uvažujme, že sčítáme body R a P+Q. Tyto body mají stejnou souřadnici x a je tady výsledek toho součtu $\mathcal{O}$.[18] Algebraicky lze definovat výsledek součtu $P \oplus Q = (r_1, r_2)$ jako

$$r_1 = \delta^2 - p_1 - q_1$$
$$r_2 = \delta(q_1 - r_1) - q_2$$
$$\delta = \frac{p_2 - q_2}{p_1 - q_1}$$

O δ lze tedy hovořit jako o směrnici přímky spojující body P a Q.

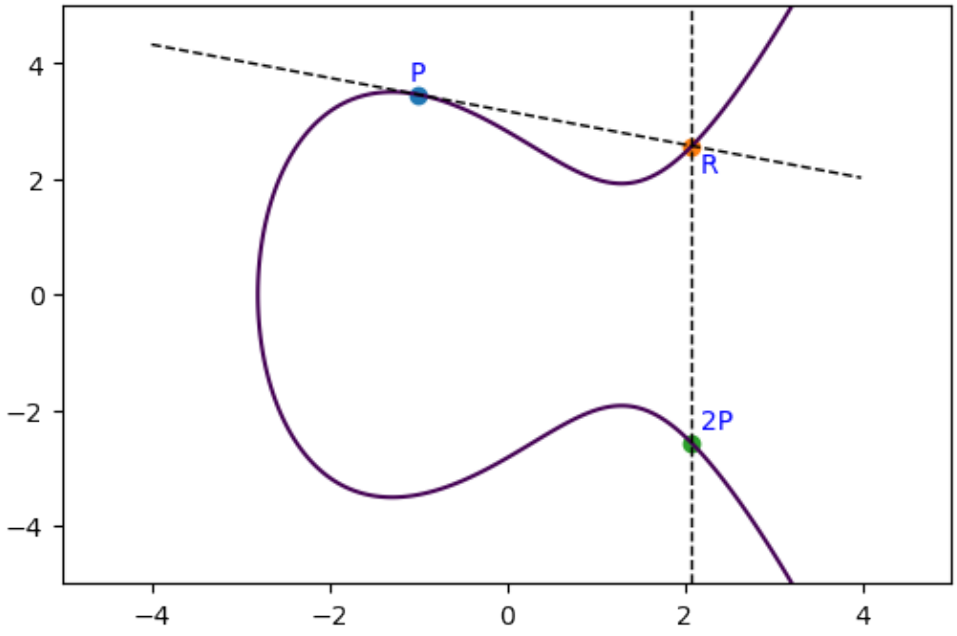


Obrázek 2.4: Součet bodů na křivce

Situace se ale liší, pokud sčítáme bod P sám se sebou. Neboli $P \oplus P$ nebo také značíme jako 2P. Geometrická představa je ta, že se vytvoří tečna k bodu P a jiný průsečík s křivkou E než bod P je opět náš bod R. Ten zrcadlíme podle osy x a tam vznikne bod 2P. Tento proces je znázorněn na obrázku 2.5. [18][19][5]

Algebraicky je možné bod $P \oplus P = (r_1, r_2)$ spočítat z těchto rovnic. [18]

$$r_1 = \delta^2 - 2p_1$$
$$r_2 = \delta(p_1 - r_1) - p_2$$
$$\delta = \frac{3p_1^2 + a}{2p_2}$$



Obrázek 2.5: Sčítání dvou stejných bodů

Výjimka nastává, pokud y souřadnice bodu je rovna 0. Poté tečna nemá žádný průnik s E a výsledek je tedy $\mathcal{O}$.

Jak již bylo řečeno, definované sčítání a přidání bodu $\mathcal{O}$ tvoří abelovskou grupu. Splňuje tedy následující vlastnosti.

- $P \oplus \mathcal{O} = \mathcal{O} \oplus P = P$
- $\forall P \in E,\ P = (p_1, p_2),\ \exists\ P^{-1} \in E,\ P \oplus P^{-1} = \mathcal{O}$
- $\forall P, Q, S,\ (P \oplus Q) \oplus S = P \oplus (Q \oplus S)$.
- $\forall P, Q,\ P \oplus Q = Q \oplus P$

Pro hledání inverzního bodu P^{-1} k bodu $P = (p_1, p_2)$ stačí souřadnici p_2 vynásobit hodnotou -1 a tím bod P zrcadlit podle osy x. Tedy $P^{-1} = (p_1, -p_2)$.[18]

2.4 Eliptické křivky nad Galoisovými tělesy

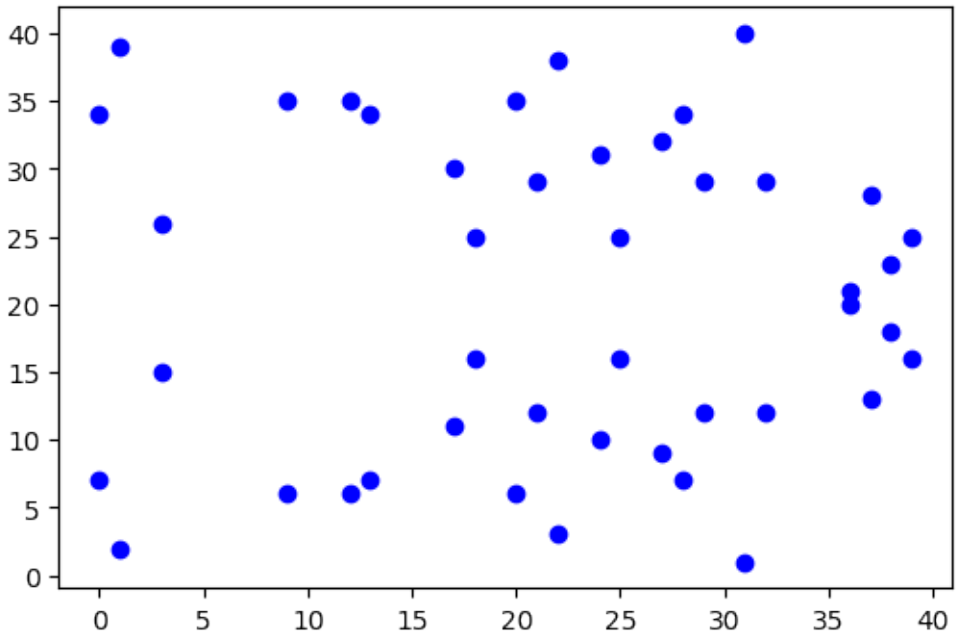
V oboru kryptografie se používají jenom eliptické křivky nad GF a to kvůli rychlejšímu a přesnému počítání nad celými čísly. Definice eliptické křivky nad Galoisovými tělesy (GF) je velice podobná té nad reálnými čísly.[20]

Definice 3 (Eliptická křivka nad $GF(p^k)$) *Eliptická křivka nad $GF(p^k)$, kde p je prvočíslo je množina*

$$E(GF(p^k) = \left\{(x, y) \mid x, y \in GF(p^k), y^2 = x^3 + ax + b\right\}$$

kde $a, b \in GF(p^k)$ splňující $4a^2 + 27b^2 \neq 0$ a do definované množiny přidáme bod $\mathcal{O}$.

Všechny operace a rovnice definované v podkapitole 2.2 a 2.3 nad GF jsou stejné. Jen počet bodů je na křivce konečný a operace sčítání a násobení se provádí modulo p.[20] Na obrázku 2.6 jsou zobrazeny body na křivce $y^2 = x^3 - 5x^2 + 8$ kde $x, y \in GF(41)$. Eliptická křivka nad GF spolu se sčítáním definovaným v podkapitole 2.3 tvoří opět abelovskou grupu.[20]



Obrázek 2.6: $y^2 = x^3 - 5x^2 + 8$ *mod* 41

2.5 Problém diskrétního logaritmu nad EC

Body na eliptické křivce s bodem $\mathcal{O}$ a definovaným sčítáním tvoří abelovskou grupu. Můžeme tedy vzít libovolný bod P a vynásobit ho celým číslem n. To je to samé, jako n-krát P sečíst.

$$n \in \mathbb{N},\ nP = P \oplus P \oplus ... \oplus P$$

Nejmenší r kde $r \in \mathbb{N}$ pro které platí, že rP = P je řádem bodu P. Toto platí, protože bodů je konečně mnoho, a proto musí nastat moment, kdy (r-1)P = $\mathcal{O}$. A poté již triviálně platí, že rP = P. Tedy P je součástí cyklické grupy.

Tímto násobením číslem n díky uzavřenosti grupy opět získáme nějaký bod Q na křivce E. Pokud známe body P a Q, je k najití čísla n potřeba vyřešit problém diskrétního logaritmu nad EC. Na tento problém neexistuje lepší řešení než algoritmus Pollardovy rho metody o $O\sqrt{\frac{\pi n}{2}}$ krocích. [5][18][20]

2.6 Generování soukromého a veřejného klíče

Pro generování páru soukromého a veřejného klíče je nutné dodržet následující postup. Nejprve je potřeba vybrat křivku s následujícími parametry.

- modul p
- koeficienty křivky a, b
- bod $G \in E$ řádu q a jenž je generátorem cyklické grupy

Dále zvolíme náhodné číslo d pro které platí $1 < d < q$. Vypočítáme bod P=dG a definujeme soukromý a veřejný klíč.

$$SK = \{d\}$$

$$VK = \{p, a, b, q, G, P\}$$

Abychom dokázali jako útočník spočítat soukromý klíč, musíme vyřešit P / G = d. Tedy již definovaný problém diskrétního logaritmu na EC. To je pro vhodně zvolenou křivku se správnými parametry výpočetně nezvládnutelný úkol. Je vhodné používat již ověřené křivky a neexperimentovat s vlastním nastavením parametrů.[18]

2.6.1 Double-and-add algoritmus

Otázkou ale nastává, jak je možné, že operace P=dG je výpočetně zvládnutelná. Kdyby se tento výpočet prováděl naivně pouze postupným sčítám bodu G d-krát, nebyli bychom na tom s náročností problému lépe než útočník. Příhodně existuje algoritmus double-and-add, který tento výraz spočítá ve $\log_2 d$ operací.[18][20]

V algoritmu jako první převedeme d do binárního zápisu

$$d = d_0 + 2d_1 + 2^2 d_2 + ... + 2^r n_r$$

kde $d_0, ..., d_r \in \{0, 1\}$. Následně spočítáme

$$dP = d_0 Q_0 + d_1 Q_1 + ... + d_r Q_r$$

kde $Q_i = 2^i P$ pro i = 0, 1,..., r. V algoritmu se provede pouze r operací sčítání bodů na křivce. Díky tomu je generování klíčů výpočetně zvládnutelné. [20]

2.7 Podpis na eliptických křivkách

Důležitým prvkem kryptoměn je digitální podpis, jak již bylo řečeno, ten zaručuje nepopiratelnost, integritu a autentizaci. Algoritmus pro podepisování zpráv je probírán v této podkapitole.

Mějme soukromý klíč respektive veřejný klíč definovaný množinou {d} respektive {p, a, b, q, G, P}. Podpis na eliptických křivkách (ECDSA) je uspořádaná dvojice čísel (r, s). Každé z nich má bitovou délku stejnou jako q. Při generování podpisu pro zprávu x postupujeme následovně.

- náhodně vybereme celé číslo k_E, kde $0 < k_E < q$
- spočítáme bod $R = k_E G$, r označíme x-ovou souřadnici bodu R
- vypočítáme podpis $s \equiv (hash(x) + dr))k_E^{-1}\ mod\ q$

Takto jsme získali podpis s. Pro ověření podpisu je potřeba si uvědomit, že rovnici $s \equiv (hash(x) + dr))k_E^{-1}\ mod\ q$ je možné upravit na

$$k_E \equiv s^{-1}hash(x) + ds^{-1}r\ mod\ q$$

Celou rovnici vynásobíme bodem G

$$Gk_E \equiv Gs^{-1}hash(x) + dGs^{-1}r\ mod\ q$$

Již víme, že dG=P, proto to nahradíme i v rovnici

$$Gk_E \equiv Gs^{-1}hash(x) + Ps^{-1}r\ mod\ q$$

V této rovnici se již nevyskytuje soukromý klíč d a tedy ten, kdo podpis ověřuje, může spočítat levou i pravou stranu rovnice. Vyjdou dva body. Pokud tyto body mají stejnou souřadnici x, je podpis validní. Jedině ten, kdo znal soukromý klíč d, mohl tento podpis vytvořit.

2.7.1 Vhodné použití eliptických křivek

Při použití EC je vždy třeba dbát na vhodně zvolené parametry. Například NIST vydal seznam bezpečných křivek s parametry, které je bezpečné používat[6].

Například Bitcoin i Ethereum v současné době používají křivku secp256k1[5], která má následující parametry.

- p = FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFE FFFFFC2F
- a = 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

[6] `https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-186-draft.pdf`

- 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000007
- G = 02 79BE667E F9DCBBAC 55A06295 CE870B07 029BFCDB 2DCE-28D9 59F2815B 16F81798
- n = FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFE BAAEDCE6 AF-48A03B BFD25E8C D0364141

Parametr G je napsán v komprimovaném formátu. Ten se získá tak, že se uvede jen x-ová souřadnice bodu G. Y-ová se již snadno dopočítá z rovnice eliptické křivky a ušetří se tímto způsobem místo. Je potřeba si uvědomit, že v rovnici je y^2, proto se v komprimovaném formátu musí přidat bit, který říká, jestli je bod nad osou x nebo pod ní. Neboli jestli je y-ová souřadnice kladná nebo záporná.[18]

Eliptické křivky se používají v kryptoměnách, protože mají výrazně bitově kratší klíče než například RSA. Vzhledem k tomu, že přímo do blockchainu se ukládá veřejný klíč, je délka veřejného klíče důležitý parametr. Tabulka 2.1 ukazuje různé bitové délky klíčů, které jsou bezpečnostně ekvivalentní. EC mají násobně kratší hodnotu klíče než RSA. [21]

Tabulka 2.1: Porovnání bitové délky klíčů při podobném zabezpečení

RSA	**ECC**
3072	256
15360	512

Kapitola 3

Ethereum

Bitcoin dopodrobna popsaný v kapitole 1 funguje na důležitých konceptech jako jsou blockchain, dosažení konsenzu v rámci všech nodů algoritmem POW, zabezpečení Bitcoinu pomocí asymetrické kryptografie a další. Tyto koncepty jsou používané i v kryptoměně Ethereum, mohou se však lišit v technických detailech. Ethereum není protokol, který se používá pouze k transakci měny, je to mnohem komplexnější mechanizmus. Do blockchainu lze ukládat programy psané v Turingově úplném jazyce a ty spouštět. Tato funkcionalita umožňuje psaní decentralizovaných aplikací, které je možné použít například na finanční operace nebo vydávání a spravování tokenů. V této kapitole se budeme věnovat konceptům, které umožňují realizaci protokolu Etherea.

3.1 Obecný pohled na Ethereum

Bitcoin je protokol, který pracuje pouze s měnou bitcoin. Všechny nody si drží kopii blockchainu a uživatel může tuto kopii změnit tím, že udělá broadcast transakce a nody transakci vloží do bloku, který následně mineři vytěží. Bicoinový blockchain je tedy záznam všech transakcí, které mají pouze možnost přeposílat bitcoin. Tento blockchain si můžeme představit jako veřejnou účetní knihu.

Tato analogie však nelze vztáhnout na Ethereum. To totiž nepracuje pouze se svojí měnou, která se nazývá ether (ETH), v rámci transakcí. Ethereum je spíše než účetní kniha jednovláknový stavový počítač.[22][23] Všechny nody si drží kopii globálního stavu počítače a mohou spustit kód, pokud o to uživatel požádá. Kód je uložen v rámci globálního stavu, který je uložen do blockchainu. Když chce uživatel spustit kód, musí vytvořit transakci, která má jako příjemce adresu kódu, který se má spustit. Mineři transakce vloží do bloku, ten mineři vytěží, ale součástí vytěžení není pouze vyřešení POW algoritmu, ale také vykonání operací, které jsou definované v kódu. Toto spuštění změní globální stav jednovláknového počítače, na kterém se všechny nody shodnou

podle konsenzu a výsledný globální stav se uloží do blockchainu. Tento počítač se nazývá Ethereum Virtual Machine (EVM).[22][23][24]

3.2 Účty

Etherový účet je entita, která má etherový zůstatek a může posílat transakce do protokolu Etherea. Entita je uložena do blockchainu a postupem času se hodnoty (například zůstatek etheru), které má v sobě uložené, mění. Bitcoin má v blockchainu uložené jen historie transakcí a zůstatky k jednotlivým adresám je potřeba z historie dopočítat. Účty mohou být ovládané pomocí uživatelů (vnější účty) nebo druhým typem účtů jsou programy (smart kontrakty) nahrané do blockchainu. Smart kontrakty jsou to, čím je Ethereum zajímavé. Jsou v nich definované výpočty, které transakce spouštějí. [22][25][24]

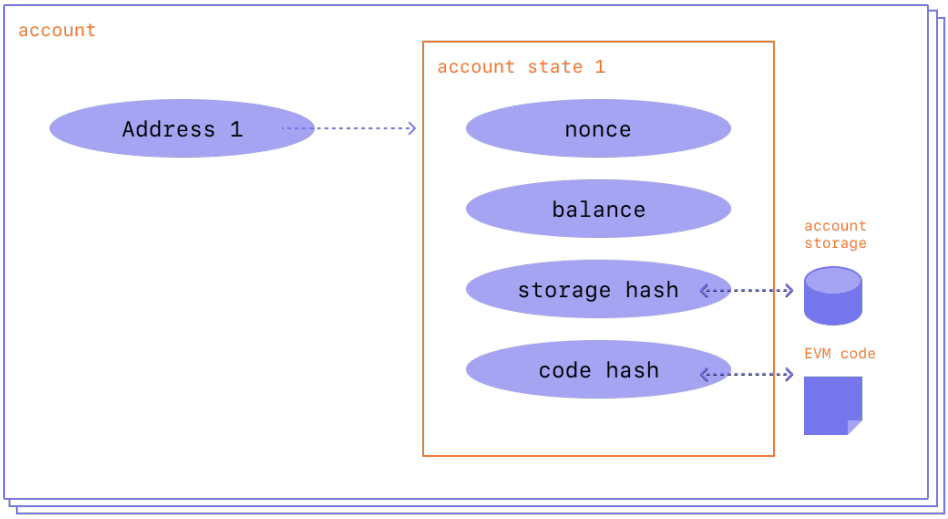
3.2.1 Vnější účty

Vnější účet je koncepčně to samé, jako je Bitcoinová adresa a uživatel, který vlastní soukromý klíč ze kterého je Bitcoinová adresa odvozena. Vnější účet může vytvářet transakce, které podepíše svým soukromým klíčem a disponuje zůstatkem etheru. Transakce mezi vnějšími účty slouží pouze pro přenos etheru. Pokud by existovaly pouze vnější účty, tak by Ethereum bylo funkčností stejné jako Bitcoin. [25][24]

Vnější účty jsou identifikovatelné pomocí jejich adresy. Ta se stejně jako u Bitcoinové adresy odvodí ze soukromého klíče. Ze soukromého klíče se odvodí veřejný klíč a tento veřejný klíč se zahashuje pomocí hashovací funkce Keccak-256. Poté se vezme spodních 20 bytů a to tvoří adresu vnějšího účtu. [25]

3.2.2 Smart kontrakty

Neexistují jen vnější účty, Ethereum používá ještě jeden typ účtů, které přináší inovaci oproti Bitcoinu. Smart kontrakty jsou kusy kódu nahrané do blockchainu a tedy nejde jejich obsah poté měnit, stejně jako nejde změnit historie transakcí u Bitcoinu. Samy od sebe nemohou spustit svůj kód a mohou vytvořit transakci pouze v případě, že na adresu smart kontraktu přišla transakce. Transakce učená adrese smart kontraktu spustí kód, který vykoná výpočet na EVM. Nahrání smart kontraktu do blockchain může provést vnější účet nebo jiný smart kontrakt, za toto nahrání se ovšem platí poplatky. Smart kontrakty jsou také identifikované pomocí adresy. Ta se vytvoří zahashováním adresy toho, kdo smart kontrakt vytvořil, a počtu transakcí, které byly z adresy stvořitele odeslány. [25]



Obrázek 3.1: Rozložení účtu [1]

Účet jak vnější tak smart kontrakt je definován následující čtveřicí údajů.[24]

- nonce: Čítač, který se inkrementuje v momentě, kdy se pošle transakce. Node nevloží do bloku transakci, pokud nonce není o jedna větší než nonce poslední transakce z dané adresy. Nonce slouží k tomu, aby transakce byla skutečně povedena pouze jednou.

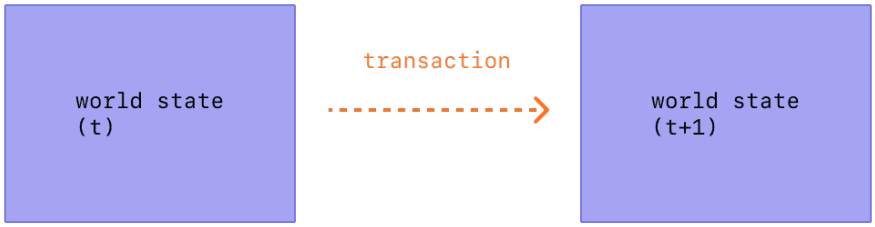
- zůstatek: Počet ETH, kterým účet disponuje. Zůstatek se udává ve wei. 1 ETH je 10^{18} wei.

- hash kódu: Hash kódu podle kterého se odkazuje na kód smart kontraktu v EVM. Nody u sebe drží v databázi kód a pokud přijde transakce, tak ho spustí. Tato položka se jako jediná z uvedených nikdy nezmění. Vnější účty mají zde uloženou hodnotu hashe prázdného řetězce, protože vnější účet není smart kontrakt, takže žádný kód nemá.

- kořen stavu: 256 bitový kořen Merkle-Patricia stromu (MPT). Ten odkazuje do databáze, kde je uloženo mapování hashe klíče a hodnoty. Slouží k uložení všech dat, které smart kontrakt potřebuje. Například hodnoty proměnných ve smart kontraktu. Veřejný účet má zde uložen hash prázdného řetězce.

3.3 Transakce

Transakce je operace podepsaná soukromým klíčem, která pochází z vnějšího účtu. Základní operace je přesun ETH z jednoho veřejného účtu na druhý. Tím možnosti transakce nekončí, dále může nahrávat smart kontrakty do blockchainu a spouštět smart kontrakty. Transakce vždy změní globální stav na kterém se musí všechny nody shodnout.



Obrázek 3.2: Transakce mění globální stav [1]

Každá transakce obsahuje následující položky. [26][24]

- příjemce: Adresa toho, komu je transakce určena.
- podpis: Pro identifikaci odesílatele a pro další důležité kryptografické vlastnosti digitálního podpisu diskutovaného v kapitole 2.
- částka: Kolik ETH se má na adresu příjemce poslat. Udává se ve wei.
- spropitné: Odměna tomu, kdo transakci vložil od bloku.
- data: Zde se udávají potřebná data pro transakci. Toto pole není povinné a jeho důležitost vysvětlíme v následujícím odstavci.
- limit gasu: Kolik maximálně gasu je možné pro tuto transakci spotřebovat. Gas je do detailu popsán v podkapitole 3.6. Stručně řečené je gas platidlo za provádění operací při spuštění kódu. Gas se kupuje za ETH.
- maximální cena gasu: Kolik je uživatel ochoten maximálně zaplatit za jednotku gasu.

Transakce obsahuje položku data, do které se v případě vytváření smart kontraktu ukládá zkompilovaný kód, který definuje kontrakt. Při vytváření nového smart kontraktu je potřeba speciální transakce, která má adresu příjemce 0x0. [22] Poté node ví, že se jedná o nahrání smart kontraktu do blockchainu.

V případě, že adresa příjemce je adresa smart kontraktu, tak položka data obsahuje informace pro spuštění smart kontraktu. Například parametry a jméno funkce, která se má spustit.[22]

3.4 Blok

Blok má tu samou funkci, jakou má blok v protokolu Bitcoin popsán v podkapitole 1.5. Blok je tvořen transakcemi, které vybral node a poté tento blok vytěžil. Bloky jsou spojené dohromady v řetězu tak, že blok má v sobě uloženou hash hlavičky předchozího bloku. Pokud by se změnil jeden blok, poté by neseděly všechny následující bloky.[24][24]

Hlavička bloku Etherea je podstatně odlišná od hlavičky bloku Bitcoinu. V následujícím seznamu uvádíme, co všechno hlavička bloku Etherea obsahuje.[22][24][27][28][29]

- timestamp: Časová známka která říká okamžik, kdy byl blok vytěžen, její hodnotu nastaví miner bloku.
- číslo bloku: Kolikátý to je blok od Genesis bloku.
- základní poplatek za jednotku gasu: Minimální cena za jednotku gasu, za kterou se transakce ještě dostane do bloku.
- obtížnost: Jak moc je výpočetně náročné blok vytěžit.
- celková obtížnost: Součet všech obtížností do tohoto bloku.
- hash: Hash hlavičky tohoto bloku.
- hash rodiče: Hash rodičovského bloku. Tato položka způsobuje zřetězení bloků.
- kořen MPT transakcí: Odkaz na všechny transakce, které byly v tomto bloku použity.
- kořen MPT účtu transakcí: Odkaz na účet transakcí. Zde jsou uloženy informace o transakci po jejím vykonání. Je zde uložen status transakce (jestli byla úspěšná nebo revertovaná), hash transakce, adresa smart kontraktu, limit gasu, kolik gasu se skutečně spotřebovalo, logy z transakce (takzvané eventy)...
- kořen MPT globálního stavu: Odkaz na globální stav (stav všech účtů), poté co se vykonaly všechny transakce.

- nonce: Pokud zahashuji nonce spolu s hashí bloku, musím získat odpovídající důkaz práce.
- kým byl blok vytěžen: A kdo si také připíše odměnu na vytěžení.
- odměna za blok
- součet spotřebovaného gasu
- limit gasu: Součet limitů gasu všech transakcí
- hashe uncles blocks (UB): UB jsou bloky, které byly součástí závodu o nejdelší řetězec (vzniklo rozvětvení stejně jako v Bitcoinu). Na rozdíl od Bitcoinu UB dostanou 87.5% odměny za vytěžení a bloky, které jejich hashe přidaly do bloku, zbylých 12.5%.[28]
- odměna za přidání UB do bloku
- uzavírací cena ETH za USD toho dne, kdy byl blok vytěžen

Čas, za který se blok vytěží, se pohybuje od 12 do 14 vteřin. Pokud je tento čas nižší, tak se obtížnost POW zvýší. [27]

Velikost bloku není závislá na počtu transakcí, ale na celkovém součtu gasu limitu všech transakcí. V době psaní této diplomové práce je cíl, aby bloky měly limit 15 miliónů gasu. Je množné mít bloky až do 30 miliónů, ale to způsobí, že pro příští blok se cena gasu zvýší, a tím se by se měla snížit poptávka po transakcích.[27] Tento mechanismus je více popsaný v sekci 3.6.

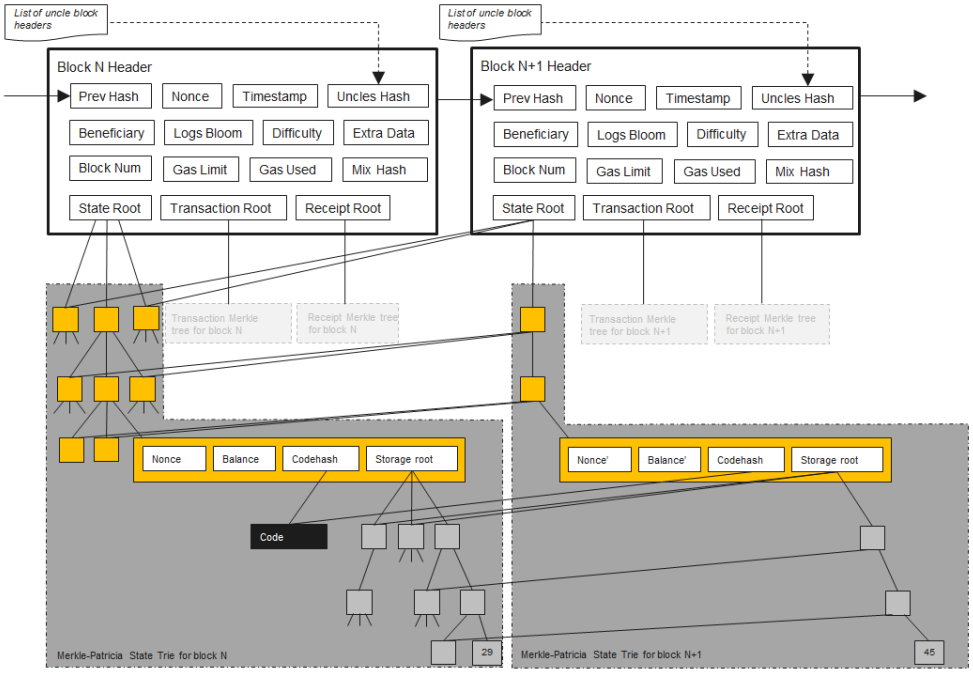
Bloky je možné si prohlížet například pomocí `https://etherscan.io/`, který je napojený na databázi fungujícího nodu a informace v něm jsou aktuální. Obrázek 3.3, který vznikl pro diskuzi na `https://ethereum.stackexchange.com`[7] názorně ukazuje, jak vypadá blockchain a co v sobě hlavičky bloku obsahují.

3.5 Stromová databáze v Ethereu

Na obrázku 3.3 je vidět, že hlavička bloku v sobě obsahuje kořen MPT. Tento kořen je odkazem do stromové databázové struktury, které se v Ethereu hojně využívá. Využívá se konkrétně pro následující data.

- Uložení globálního stavu EVM.
- Uložení transakcí které byly vykonány v bloku.
- Uložení účtů transakcí.

[7] `https://ethereum.stackexchange.com/questions/268/ethereum-block-architecture`



Obrázek 3.3: Bloky v Ethereu

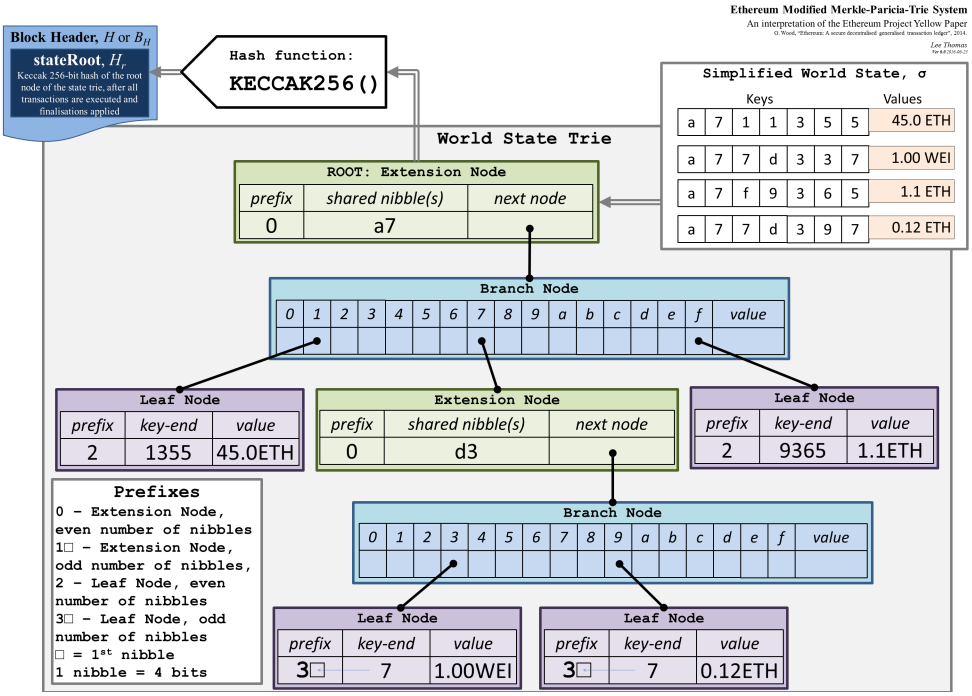
Tato databázová struktura se nazývá Merkle-Patricia tree a je v něčem podobná Merkleovu stromu popsaném v podkapitole 1.5.1.

Merkleův strom je vhodný pro SPV který ověřuje, jestli se transakce nachází v daném bloku. Merkleův strom a SPV protokol jsou popsáni v podkapitole 1.5.1. Merkleův strom v Bitcoinu předpokládá, že informace na do něj zanesené se již nebudou měnit, protože kdyby se změnily, tak by neodpovídala hodnota kořene Merkleova stromu uložená v hlavičce bloku. Není vhodný pro vyhledávání dat nebo pro měnění hodnot ve stromě. Tyto operace jsou v Ethereum časté, proto je potřeba používat jinou datovou strukturu pro uložení dat, ale je žádoucí, aby se zachovala vlastnost Merkleova stromu zajišťující integritu dat.

MPT je vhodný na mapování dat na klíč a jejich následné uložení. Klíčem může být adresa účtu a daty zůstatek na účtu nebo nonce. MPT je datová struktura, která dokáže kořen MPT rychle přepočítat po vložení, smazání a nebo pozměnění hodnoty. Další důležitou vlastností je, že nezáleží na pořadí dat, jak se do stromu vkládají, ale vždy se stromem prochází na základě hodnoty klíče. Proto pokaždé pro stejné klíče a hodnoty vnikne stejný hash kořene stromu nehledě na pořadí vložení. Kořen stromu je hodnotově závislý na datech dětí, a tak zaručí integritu dat stejně, jak to dělá Merkleův strom. [30][29]

MPT se skládá ze 4 různých vrcholů.[30]

- Prázdný vrchol, který neobsahuje žádná data.
- List stromu je klasický vrchol, který má v sobě uložený klíč a hodnotu.
- Větev je vrchol, ve kterém je 17 záznamů. Prvních 16 odpovídá hexa znakům (0-F), kterých může půl byte klíče nabývat. K nim je uložena hash vrcholu, kam záznam odkazuje. Sedmnáctý záznam je hodnota, pokud klíč končí v tomto vrcholu.
- Rozšířený vrchol má jako hodnotu hash dalšího vrcholu.



Obrázek 3.4: Merkle-Patricia tree

Obrázek 3.5, který vznikl na diskuzním fóru o Ethereu[8] velice dobře znázorňuje, jak takový MPT vypadá pro stav, kde máme 4 adresy 0xa711355, 0xa77d337, 0xa7f9365, 0xa77d397, které slouží jako klíče a k nim zůstatek na účtě v ETH. Nibble je půlka bytu a nabývá hodnot jeden hexa znak (0-F).

Všechny adresy sdílejí společný začátek adresy 0xa7, proto je v prvním rozšířeném vrcholu uložena tato část klíče a hodnota je hash větve (next node). Následující nibble se liší kromě dvou adres. Hodnoty nibblu jsou 0x1, 0x7, 0xf. Proto je ve větvi k těmto příslušným hodnotám nibblu uložena hash dalšího vrcholu.

[8] https://ethereum.stackexchange.com/questions/268/ethereum-block-architecture

Vzhledem k tomu, že dvě adresy 0xa711355 a 0xa7f8365 mají od této chvíle unikátní zbytek adresy, vzniká pro ně již pouze list stromu, kde je klíč zbytek adresy 0x1355 respektive 0x9365 a hodnoty jsou zůstatky v ETH.

Pro adresy 0xa77d337 a 0xa77d397, které nadále sdílejí hodnotu bytu 0xd3, vzniká nový rozšířený vrchol, kde uložený klíč je sdílená hodnota bytu 0xd3 a hodnota hashe dalšího vrcholu.

Tím je vrchol typu větev, která má uložené odkazy v políčkách odpovídajících hodnot 0x3 a 0x9. Ty odkazují jenom na listy, kde je uložen zbytek adresy jako klíč a odpovídající zůstatek.

Důležité je, že se vrcholy na sebe odkazují skrz hash vrcholu. To znamená, že strom zajišťuje integritu dat. Pokud by se nějaká hodnota ve stromu změnila, tak by se změnila i hash jejího vrcholu a hash kořene stromu by se změnil. MPT má tedy stejné vlastnosti jako Merkleův strom a může být i stejně využíván klienty protokolu, kteří u sebe nemají uloženou celou historii blockchainu, ale můžou ověřovat, jestli je transakce součástí bloku. Je to stejný princip, který je popsán v podkapitole 1.5.1.[30]

3.6 Gas

Na blockchain je možné nahrát smart kontrakty, které mají v sobě uložený kód. Ten je možné spustit i s parametry pomocí transakce, toto spuštění a vykonání kódu vykonávají nody. Kód obsahuje set instrukcí, které tvoří Turingův úplný jazyk, lze tedy říci, že smart kontrakty v Ethereu dokáží implementovat jakýkoliv algoritmus, který Univerzální Turingův stroj dokáže spočítat.[29][22]

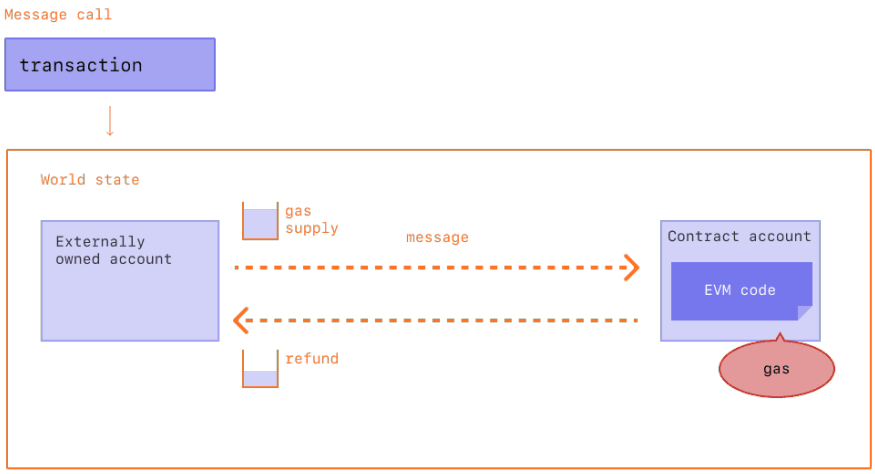
S touto vlastností je ale potřeba vyřešit takzvaný *halting problem*. Tedy jestli je možné určit, kdy se daný algoritmus zastaví. Alan Turing dokázal, že obecné řešení *halting problemu* pro libovolný kód s libovolnými vstupy není možný. [22]

Zde však nastává pro Ethereum problém. Pokud není možné vyřešit *halting problem*, jak se zabránit tomu, aby kód spuštěný nodem neběžel v nekonečné smyčce a tím způsobil denial of service (DoS).[22]

Kvůli výše zmíněnému problému byl představen gas. Gas přestavuje jednotku, která určuje, jak je která operace výpočetně náročná pro EVM. Vzhledem k tomu, že všechny transakce potřebují výpočetní výkon nodů, tak gas je poplatek za tuto službu. Čím více výpočetně nebo paměťově náročná je operace, tím více gasu je na tuto transakci potřeba vyhradit.

V podkapitole 3.3 je popsáno, že transakce v sobě obsahuje limit gasu a maximální cenu gasu za jednotku gasu. Limit znamená, kolik maximálně může transakce a případné spuštění smart kontraktu spálit gasu. Gas který se nevypotřeboval se vrací uživateli, od kterého daná transakce byla podepsána. Maximální cena za jednotku gasu říká, kolik je uživatel, který tuto transakci podepsal, ochotný maximálně zaplatit za jeden gas.[31] Obě tyto položky se udávají v gwei neboli giga-wei. Jeden gwei je $10^{-9} ETH$.

V případě, že vykonání transakce překročí gas limit nastavený v transakci, je výpočet ihned ukončen a globální stav se nezmění. Uživatel není nijak kompenzován za nedokončenou transakci.[31][22][29]



Obrázek 3.5: Pro provedení transakce je potřeba gas[1]

Každý blok v sobě obsahuje základní poplatek za jednotku gasu (base fee). Tento poplatek se dynamicky mění pro každý blok podle poptávky uživatelů po místě v bloku. Jak již bylo zmíněno, velikost bloku se může lišit. Jeho maximální velikost je 30 miliónů jednotek gasu, ale to je dvojnásobek cílené velikosti bloku, proto se základní poplatek za jednotku gasu zvýší, pokud je velikost bloku větší než cílená velikost. Může se zvýšit maximálně o 12.5% oproti ceně v předchozím bloku. V případě, že je velikost menší než 15 miliónů gasu, tak se cena analogicky snižuje.[31]

Celkový poplatek za transakci se spočítá vynásobením limitu gasu za danou transakci se součtem poplatku za jednotku gasu a spropitného.

$$fee = gasUnits * (baseFee + tip)$$

Základní poplatek (base fee) se nepřipíše minerovi, tento poplatek se vyřadí z oběhu. Miner pouze dostane spropitné. Podle výše spropitného nody často určují, jestli transakci do bloku přidají a nebo ne.

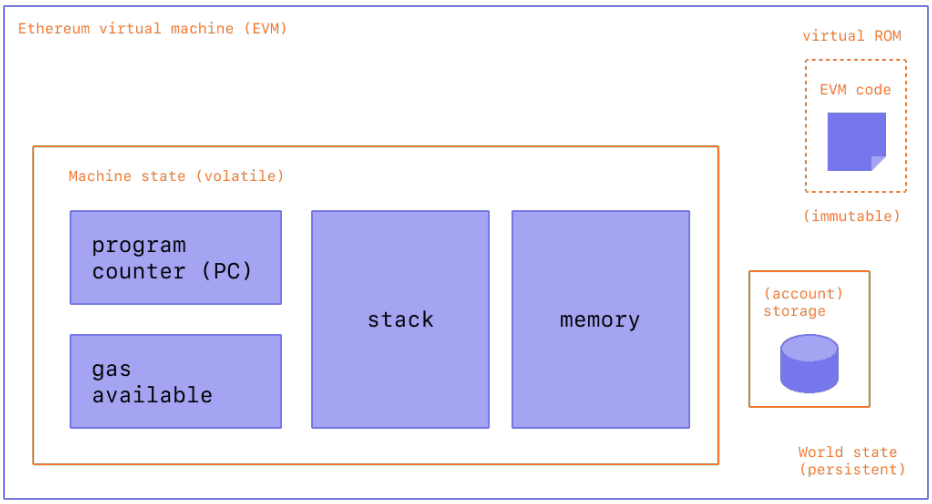
3.7 Ethereum virtual machine

EVM je globální stavový počítač, přičemž kopii tohoto stavu si každý node u sebe drží. Protokol Etherea je zodpovědný za to, že se vždy najde konsenzus

ohledně podoby globálního stavu. [32][22]

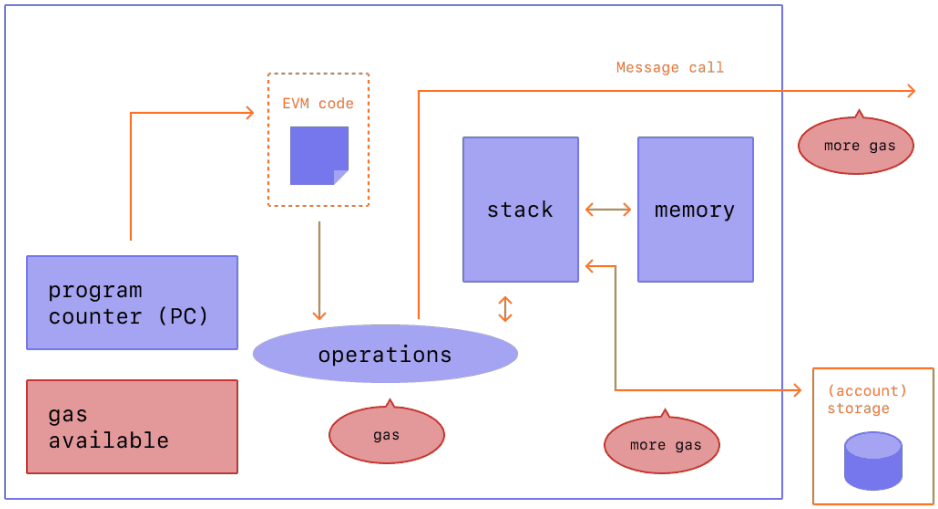
Tento globální stav si nedrží pouze zůstatky jednotlivých účtu, ale také například hodnoty uložené v proměnných v rámci kódů smart kontraktů. Instrukce, které tvoří kód, musí být zcela deterministické, aby bylo možné pro každý node dojít ke stejnému stavu. Kdyby nebyly deterministické, nebylo by možné pro nody při spuštění stejného kódu dosáhnout stejného výskledků globálního stavu a dosáhnout konsenzu na jeho podobě. [32][22]

Když se vykonává transakce, která volá smart kontrakt, tak se EVM chová jako zásobníkový počítač s maximální hloubkou pro 1028 položek. Jedna položka má velikost 256 bitů. Během vykonávání má EVM nejen zásobník, ale také paměť, která se po vykonání transakce nikam neukládá. Pro spuštění kódu je potřeba počítadlo instrukcí, aby EVM vědělo, jakou instrukci dále vykonávat a nakonec kolik gasu ještě zbývá k dispozici. EVM je ilustrován na obrázku 3.6 a spuštění kódu na EVM je zobrazeno na obrázku 3.7[32][22]



Obrázek 3.6: EVM[1]

Kód ve smart kontraktu je definovaný byte kódem. Byte kód je nízkoúrovňový jazyk, kterému každý EVM rozumí. Je možné v něm smart kontrakty přímo psát, ale nejčastěji se používají vysokoúrovňové jazyky jako Solidity nebo Vyper. Smart kontrakty psané ve vysokoúrovňovém jazyku je potřeba před nahráním do blockchainu zkompilovat do byte kódu, kterému již každá implementace EVM rozumí. Byte kód je tvořen klasickými operacemi jako ADD, SUB, MUL ale jsou i specializované instrukce pro práci s blockchainem BALANCE, ADDRESS, GASPRICE...[22][32]



Obrázek 3.7: Spuštění kódu na EVM[1]

Tabulka 3.1: Příklad bytecodů pro EVM

Jméno	Gas	Operace se zásobníkem
ADD	3	a, b
MUL	5	a, b
ADDRESS	2	.
GASPRICE	2	.

3.8 Konsenzus

Ethereum dosahuje Konsenzu nad stavem blockchainu pomocí algoritmu POW, stejně jako Bitcoin popsaný v kapitole 1. Jsou tedy opět zapotřebí mineři, kteří musí bloky vytěžit. Proces těžení a validace bloků probíhá následovně.[33]

- Uživatel podepíše transakci svým privátním klíčem.
- Uživatel podepsanou transakci odešle do celé Ethereum sítě (broadcast).
- Každý node, který tuto transakci dostane, ji přidá do své databáze nevyřízených transakcí.
- Node vytvoří nový kandidátní blok z transakcí, které má uložené ve své databázi nevyřízených transakcí. Node se snaží maximalizovat poplatky, které dostane za transakci, ale nesmí překročit limit gasu na blok.

  - Node ověří u každé transakce, jestli je validní. Dále spustí kód smart kontraktu, pokud to transakce požaduje, aktualizuje svoji kopii globálního stavu a vytvoří hlavičku bloku.
  - Začne hashovat hlavičku bloku za účelem splnění podmínky algoritmu POW.

- Nodu kterému se podařilo vyřešit POW odešle ostatním nodům kandidátní blok.

- Nody ověří POW, spustí všechny transakce v bloku a zjistí, jestli došly ke stejnému stavu jako miner (porovnáním kořene MPT). Jedině pokud vše sedí, přidají kandidátní blok na konec blockchainu a tím nový globální stav.

- Nody odstraní ze své databáze nezprocesovaných transakcí transakce použité v bloku.

3.9 Solidity

Solidity je objektově orientovaný vysokoúrovňový jazyk, který byl vytvořen pro implementaci smart kontraktů. Smart kontrakty lze definovat jako neměnné počítačové programy, které běží deterministicky v EVM, které je součástí Ethereum protokolu.[22]

Velice důležité je slovo deterministicky. To znamená, že pro každé spuštění smart kontraktu v tom samém stavu a pro ty samé vstupy musí dávat ten samý výstup. To je důležité protože, kdyby smart kontrakty nebyly deterministické, vycházel jiný globální stav pro každé spuštění a nešlo by dojít ke konsenzu pro všechny nody. Proto je pro smart kontrakty poněkud problematické například generování náhodných čísel.[22]

Jak je zmíněno v podkapitole 3.7, EVM umí spouštět byte kód. Solidity zdrojový kód musí pomocí kompilátoru *solc* přeložit do byte kódu.[22]

3.9.1 Aplication binary interface

Aplication binary interface (ABI) je interface mezi dvěma binárními moduly. Těmito moduly typicky mohou být operační systém a uživatelský program. ABI popisuje, jak jsou datové struktury a funkce přístupné ve strojovém kódu.[22]

V Ethereum ABI slouží k popsání funkcí v kontraktu. Poté co je Solidity zdrojový kód zkompilovaný, je z byte kódu velmi pracné parametry funkce, datový typ parametrů a jméno funkce získat ani nejde. Uživatel tyto informace potřebuje, aby mohl volat funkce smart kontraktu. Proto během kompilace vzniká ABI, které popisem funkcí tento problém řeší.[22]

```
// SPDX-License-Identifier: MIT
pragma solidity >=0.6.2 <0.9.0;

contract SimpleContract {
    // Adresa ktera muze posilat ETH
    address payable public owner;

    // Kontruktor, ktery take muze prijimat
    //automaticky ETH protoze je payable
    constructor() payable {
        owner = payable(msg.sender);
    }

    function deposit() public payable {
        require (msg.value > 1 ether);
    }

    function withdraw(uint amount) public {
        // Vybirana castka nesmi presahnout polovinu vkladu
        require(amount < (address(this).balance >> 2));
        (bool success, ) = owner.call{value: amount}("");
        require(success, "Failed to send Ether");
    }
}
```

Listing 3.1: Jednoduchý smart kontrakt

Vytvořili jsme ukázkový smart kontrakt, který je ukázaný ve výpisu 3.1. Program je napsán tak, že kdokoliv může do smart kontraktu vložit ETH vytvořením transakce, která má v sobě uloženo, kolik wei se má do kontraktu poslat a volá funkci *deposit*. Pouze ten, kdo kontrakt do blockchainu nahrál, může obsah smart kontraktu vybrat a to transakcí, která volá funkci *withdraw*. Je možné vybrat maximálně polovinu obsahu smart kontraktu najednou. ABI, které vznikne zkompilováním takového smart kontraktu, je ukázáno v 3.2. Jsou zde popsány vstupy do funkce, případné výstupy z funkcí, jména funkcí a podobně.

```
[
  {
    "inputs": [],
    "name": "deposit",
    "outputs": [],
    "stateMutability": "payable",
    "type": "function"
  },
  {
    "inputs": [],
    "stateMutability": "payable",
    "type": "constructor"
  },
  {
    "inputs": [
      {
```

```
17          "internalType": "uint256",
18          "name": "amount",
19          "type": "uint256"
20        }
21      ],
22      "name": "withdraw",
23      "outputs": [],
24      "stateMutability": "nonpayable",
25      "type": "function"
26    },
27    {
28      "inputs": [],
29      "name": "owner",
30      "outputs": [
31        {
32          "internalType": "address payable",
33          "name": "",
34          "type": "address"
35        }
36      ],
37      "stateMutability": "view",
38      "type": "function"
39    }
40  ]
```

Listing 3.2: ABI

Když transakce volá funkci ve smart kontraktu, tak využívá právě ABI pro sestavení dat do transakce. Transakce určí, jakou funkci chce volat a s jakými parametry. Pro volání funkce se nejprve spočítá hodnota, která určí, kterou funkci chceme volat. Ta se získá spočítáním Keccak-256 hashe signatury funkce. Signatura je jméno funkce a datový typ parametrů. Z této hashe se vezmou 4 nejvíce významné byty. Například funkce *withdraw* má signaturu *withdraw(uint256)*. 4 nejvíce významné byty jsou *0x2e1a7d4d*, těmto bytům se říká také selektor funkce. [34]

V datech transakce jsou i parametry, které transakce do funkce posílá. Například pokud by transakce, kterou podepsal majitel kontraktu (ten, kdo tento kontrakt nahrál do blockchainu), měla za úkol vybrat 1 ETH ze smart kontraktu popsaném v 3.1, tak by data transakce vypadala takto *2e1a7d4d000-0000000000000000000000000000000000000000000000001000000000000000000*. Nejprve 4 byty selektoru, následované hodnotou parametru funkce.

3.9.2 Datové typy

Solidity nabízí klasické základní datové typy jako je *interger, boolean, statické bytové pole, dynamické bytové pole, enum, pole, struktury, mapovaní (klíč který je namapován s hodnotou)*.[22]

K dipozici jsou však i datové typy, které jsou určené pro práci s kontrakty. V Solidity je k dipozici objekt *address*. Tento objekt má následující členské funkce a proměnné.[22]

- *address.balance* obsahuje zůstatek ETH ve wei na dané adrese.
- *address.transfer* je funkce, která má jako parametr částku, která se má poslat ze smart kontraktu na danou adresu. Pokud nastane chyba, funkce vyhodí výjimku.
- *address.send* je podobná funkce jako funkce *address.transfer*, jen vrací false v případě chyby.
- *address.call* určitým způsobem simuluje transakci ze smart kontraktu. Může volat funkce jiných smart kontraktů, může posílat ETH na účty. Záleží na datech, která se této funkci nastaví přes parametr.
- *address.callcode* je podobná funkce jako *address.call*. Rozdíl je, že když se použije *address.callcode*, tak volaný pracuje v kontextu volajícího. To například může znamenat, že pokud kontrakt A volá funkci B přes funkci *address.callcode*, tak kontrakt B může měnit hnodnotu proměnné v kontraktu A.
- *address.delegatecall* je obdobná jako *address.callcode*. Rozdíl je v tom, že volající má i stejný *msg.sender* a *msg.value* jako volaný.

Také můžeme využívat jednotky etheru (*wei*, *finney*, *szabo*, a *ether*) v kódu jako surfix za číslem.[22]

3.9.3 Předdefinované globální proměnné a funkce

Kontrakt spuštěný v EVM má k dispozici několik globálních objektů, které může používat. Jsou to například *msg*, *blocks* a *tx*.

Objekt *msg* obsahuje informace o transakci, která volala daný smart kontrakt. Najdeme tam tyto členské atributy.[22]

- *msg.sender* obsahuje adresu toho, kdo daný kontrakt zavolal. Kontrakt může volat jiný smart kontrakt a poté *msg.sender* obsahuje adresu volajícího smart kontraktu.
- *msg.value* je hodnota ETH, která je do kontraktu posílaná ve wei.
- *msg.gas* udává, kolik gasu zbývá během spuštění.
- *msg.data* říká, jaká data byla do kontraktu poslána transakcí.
- *msg.sig* obsahuje selektor funkce, která je volaná.

Další objektem je *tx*, což je kontext transakce.[22]

- *tx.gasprice* je cena gasu pro danou transakcí.
- *tx.origin* je adresa vnějšího účtu který vytvořil transakci.

Nakonec je k dispozici objekt *block*, který představuje kontext bloku.[22]

- *block.blockhash* je členská funkce, které přijímá jako vstupní parametr číslo bloku a vrací hash tohoto bloku.
- *block.coinbase* je adresa toho, kdo si připisuje odměnu za vytěžení bloku.
- *block.difficulty* říká obtížnost POW algoritmu pro současný blok.
- *block.gaslimit* je hodnota gas limitu v současném bloku.
- *block.number* je vzdálenost současného bloku od Genesis bloku.
- *block.timestamp* obsahuje časovou známku, kterou miner nastavil jako čas, kdy byl současný blok vytěžen.

Vestavěné funkce, které je možné v Solidity používat, jsou uvedené v následujícím seznamu.[22]

- *addmod, mulmod* jsou funkce pro modulární sčítání a násobení.
- *keccak256, sha256, sha3, ripemd160* jsou hashovací funkce, které se často v Ethereu používají.
- *ercecover* získá adresu z digitálního podpisu.
- *selfdestrunct* smaže současný smart kontrakt a zbytek ETH pošle na adresu danou v parametru.
- *this* vrací adresu současného kontraktu.

3.9.4 Definování smart kontraktu

V této podkapitole popíšeme, jak je možné v Solidity definovat smart kontrakt, interface, knihovnu a v čem se tyto různé přístupy k implementaci smart kontraktu liší.

Na definování smart kontraktu v Solidity se používá klíčové slovo *contract*. To je použito v ukázkovém smart kontraktu, který jsme ukázali ve výpisu 3.1 Tím se definuje objekt smart kontraktu (něco jako třída v objektově orientovaných jazycích). Dále je možné definovat *interface*, který je to samé jako *contract*, jen nejsou implementované funkce, ty je potřeba implementovat v kontraktu, který z interfacu dědí. Polední možností je *library*. Knihovna se nahraje na blockchain a slouží pouze k tomu, aby ji volaly ostatní smart kontrakty pomocí funkce *delegatecall* a využívaly implementací funkcí této knihovny.[22]

3.9.5 Funkce

Funkce v Solidity plní stejnou funkci, jako v jakémkoliv jiném programovacím jazyce. Syntaxe v Solidity pro definování funkcí je tato.

function jmenoFunkce([parametry]) public/private/internal/external [pure/constant/view/payable] [modifiery] [returns (datový typ)]

Jméno funkce a parametry jsou za sebe mluvící a dále uvádíme význam ostatním klíčových slov.

- *public* je viditelnost funkce. Defaultně jsou všechny funkce veřejné a znamená to, že funkci mohou volat všechny účty.
- *private* způsobuje, že funkci můžou volat pouze funkce definované v současném kontraktu.
- *internal* je to samé jako *private*, ale navíc funkci může volat kontrakt, který dědí z daného kontraktu.
- *external* způsobuje, že tyto funkce nemohou být volané ze současného kontraktu jinak než za pomoci klíčového slova *this*.
- *constant a view* říkají, že funkce nemůže nic měnit na stavu kontraktu.
- *pure* funkce nepracuje s členskými proměnnými, ale pouze s argumenty, které dostane při zavolání.
- *payable* je funkce, která umí přijmout ETH, který je uveden v transakci.

Fallback je speciální funkce, která nemá definované jméno. Zavolá se pokaždé, když transakce nemá žádná data nebo selektor funkce neodpovídá žádné funkci v kontraktu.[22]

3.9.6 Konstruktor a selfdestruktor

Ve chvíli, kdy je kontrakt nahráván na blockchain, spustí node kontrakt v EVM a vykoná se konstruktor. Stav kontraktu po provedení konstruktoru se uloží do blockchainu. Konstruktor se definuje pomocí klíčového slova *constructor*.[22]

Na druhou stranu selfdestuktorem končí životní cyklus smart kontraktu. Pokud je tato funkce v kontraktu zavolána, je kód smart kontrakt smazán a nelze již smart kontrakt používat.[22]

3.9.7 Eventy

Pokud je transakce dokončena, vznikne účet transakce (transaction receipt) který jíž je zmíněn v podkapitole 3.4. Tento účet obsahuje logy, které byly během vykonávání transakce vytvořeny. Eventy jsou objekty v Solidity, které logy vytváří.

Tyto logy mohou obsahovat informace, na které čekají decentralizované aplikace a na jejichž základě, dělají nějakou činnost.

Kapitola 4

Rozšiřování Etherea

V kapitole 3 je popsán protokol Etherea. A jak je používán v době psaní této diplomové práce. Jedná se však o rychle se měnící proces a jsou naplánované změny, které Ethereum zásadně změní.

Ethereum je populární blockchain. To způsobuje, že je problém s kapacitou a cena gasu je v některých chvílích velice vysoká a na zpracování transakce se musí dlouho čekat. Aby se tento problém vyřešil, je v budoucnu naplánované rozšíření kapacit Etherea a tím snížení poplatků za transakce a zvýšení počtu zpracovaných transakcí za vteřinu. V tuto chvíli Ethereum zvládne pouze přibližně 15 transakcí za sekundu, což je nedostatečné. [22]

4.1 Ethereum 2.0

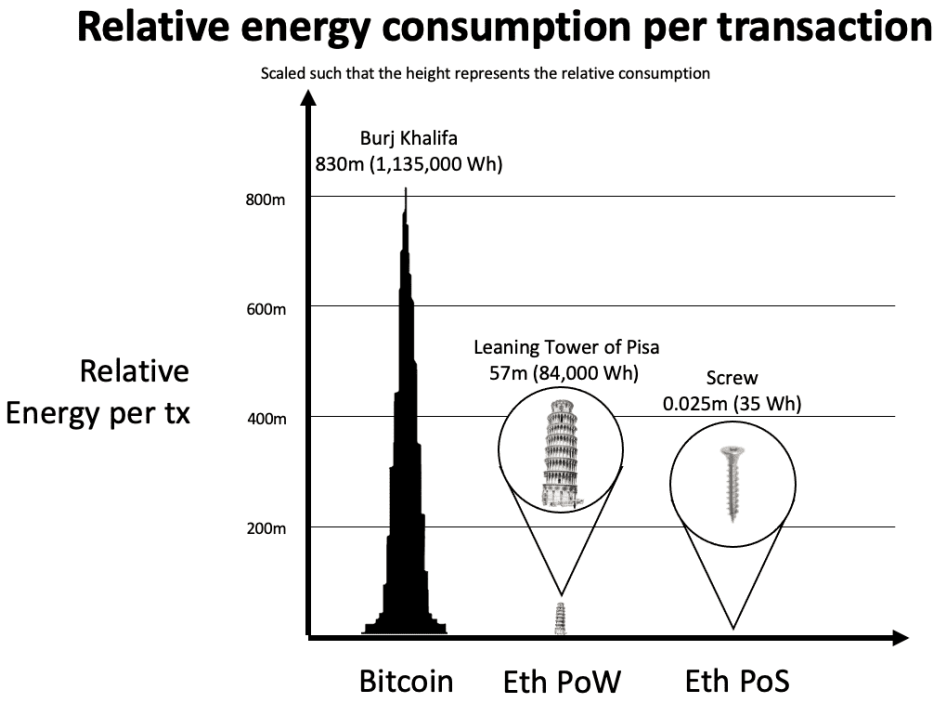
Ethereum 2.0 chce nejenom zvětšit kapacitu Etherea, ale plánuje i změnit algoritmus pro ustanovení konsenzu. Zavádí algoritmus proof of stake (POS), který má nahradit POW.

4.1.1 Proof of stake

POS již nepoužívá minery, kteří vykonávají výpočetně náročnou práci, tak jako tomu je u POW. Mineři jsou nahrazeni validátory. Validátoři jsou zodpovědní za to samé jako mineři, tedy tvoří bloky, kontrolují správnost transakcí a ukládají bloky do blockchainu.[3]

Aby se uživatel mohl stát validátorem, musí učinit speciální transakci, ve které uzamkne jako deposit 32ETH, a tímto se stane validátorem. Pokud validátor správně validuje a vytváří bloky, je odměněn spropitným stejně jako tomu bylo u POW. Pokud ale bude offline a nebo nebude validovat podle protokolu, bude mu odebírán ETH z jeho depozitu. Tím se zajistí motivace, aby validátoři pracovali poctivě. Validátoři, kteří budou kontrolovat nové bloky, jsou vybíráni náhodně, stejně tak jako validátor, který nový blok vytvoří.[3]

Výhodou POS je, že neprobíhají závody, kdo rychleji vyřeší POW a proto není tolik náročný na elektrickou energii a je ekologičtější. Porovnání poměru spotřeby energie na transakci zobrazuje obrázek 4.1. Také nody nepotřebují speciální hardware jako to je u Bitcoinových těžících farem. POS je v reálném světě více decentralizovaný, protože se může připojit každý i s málo výkonným počítačem a neměly by vznikat velké těžící skupiny, které budou mít velké procentuální zastoupení v celkovém těžení, aby se zajistil stálý příjem (v Bitcoinu odměnu za vytěžení dostane jenom ten, kdo blok vytěžil jako první). Další výhodou je bezpečné přidání dalších blockchainů (sharů) o tom ale v podkapitole 4.1.2. Nevýhodou je, že POS není tolik časem ověřený jako POW a tedy může mít nějaké nečekané slabiny. [3]



Obrázek 4.1: Porovnáni spotřeby energie na transakci[2]

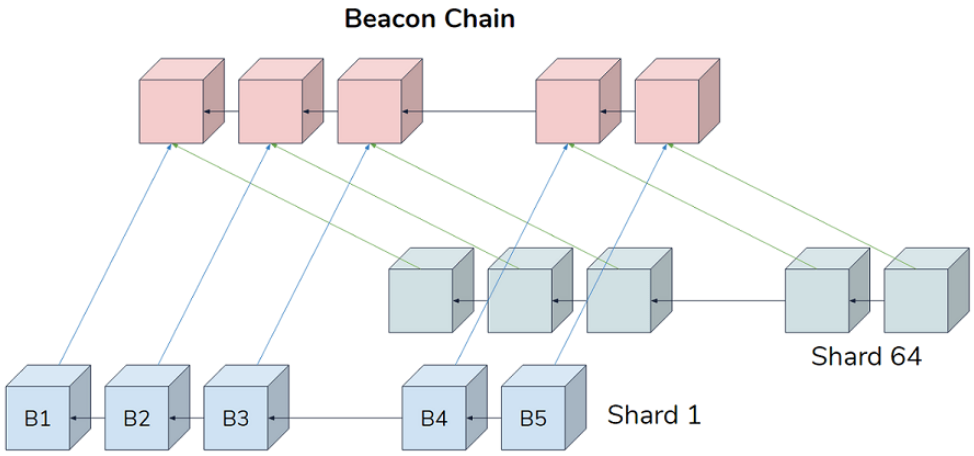
4.1.2 Beacon chain a sharding

Beacon chain s shardy je rozšíření původního blockchainu Etherea pro větší kapacitu, rychlost a snížení poplatků za transakci. Problém současného návrhu je, že každý node musí vykonat každou transakci, aby ověřil správnost globálního stavu. Aby se tento proces zrychlil, musí se buď zvýšit kapacita nodů, a nebo

přidat nody, které nebudou dělat stejnou práci, ale rozdělí si ji. V této podkapitole je popsán ten druhý přístup.[3][35]

Nyní je v plánu vytvořit 64 shardů, což jsou separátní blockchainy, které potřebují, aby jim byli přiřazení validátoři a zprostředkovatele dat mezi jednotlivými shardy. O to se stará Beacon chain, který přiděluje validátory shardům, zná globální stav všech shardů a dává tyto stavy k dispozici ostatním shardům. Beacon chain je v tuto chvíli spuštěn a testuje se jeho funkčnost.[3]

Jeden shard v sobě obsahuje pouze část dat z celého protokolu. Budou tedy menší hardwarové nároky na spuštění nodu, a přitáhne to tedy více účastníků do protokolu a způsobí větší decentralizovanost a bezpečnost protokolu.[35]



Obrázek 4.2: Beacon chain a sharding[3]

Beacon chain je něco jako dirigent celého protokolu, který udává tempo. Jeho tempo se skládá ze slotů. Slot je chvíle, kdy shardy a Beacon chain přidají nové bloky. Slot trvá 12 vteřin a epoch se skládá z 32 slotů. [35]

Beacon chain primárně registruje adresy validátorů a jejich zůstatky ETH. Dále náhodně přiřazuje validátory shardům a drží odkazy na bloky přidané všemi shardy.[35][3]

Každý slot se vytvoří komise pro shardy a Beacon chain, která je tvořená minimálně ze 128 validátorů. Tato komise hlasuje pro nový blok navržený navrhovatelem, který se náhodně zvolí pro každý shard Beacon chainem. Po každém slotu je vytvořena nová náhodná komise.[3]

Níže popisujeme tři fáze, ve kterých se má přejít z Etherea na Ethereum 2.0.

- Fáze 0 se nazývá „The Beacon Chain“ a přináší nový blockchain zvaný

Beacon chain, který bude mít POS protokol a možnost dát uživatelům jako deposit svých 32ETH, aby se z nich stali validátoři. Fáze 0 v tuto chvíli probíhá a Beacon chain je spuštěn. [35]

- Fáze 1 je „The Merge“. Současný blockchain Etherea se spojí s Beacon chainem. Bude to znamenat konec POW a v Ethereu se bude používat pouze POS. Současný blockhain se pravděpodobně stane shardem a bude používat POS protokol. Fáze 1 je v plánu spustit v roce 2022, ale datum se stále odsouvá a je možné, že se tak stane až v roce 2023.[36]
- Fáze 2 „Shard chains“ je přidání shardů do protokolu. V první fázi budou pravděpodobně shardy umět jen ukládat data, ale je možné, že pokud se to ukáže jako potřebné, budou umět spouštět nody v shardech i smart kontrakty. To je stále v diskuzi. Fáze 2 by měla být spuštěna v roce 2023. [37]

4.2 Layer 2

Layer 2 je další řešení, jak umožnit větší počet transakcí než je v současném protokolu možné. Jde o snahu přenést co nejvíce práce (například zpracování transakcí) mimo blockchain, a pak do hlavního blockchainu nahrát důkaz, že tato práce proběhla. Toho je možné dosáhnout pomocí tak zvaných rollups a mechanizmem kanálů.

4.2.1 Rollups

Rollups je jedním z řešení spadající pod Layer 2. Rollups fungují tak, že spustí více transakcí mimo hlavní blockchain, a pak se nahrají data, která z těchto transakcí vznikla na hlavní blockchain. Transakce jsou spuštěné na odděleném blockchainu a mohou mít i vlastní EVM pro spuštění smart kontraktů. Aby se ušetřil gas při nahrávání na hlavní blockchain, tak se data z transakcí spojí dohromady (dalo by se říci, že do bloku) a poté jsou nahrána na hlavní blockchain v rámci jediné transakce.[3]

Ethereum nijak neřeší, jestli jsou nahraná data validní. To musejí dělat samy implementace rollupů přes smart kontrakty nahrané do Etherea. Tyto smart kontrakty jsou zodpovědné za vložení vkladů, jeho vybrání a validovaní důkazu, jestli byla transakce správně vykonána. [3]

4.2.2 Kanály

Tento mechanizmus se hodí v případě, že víme, s kým bude výměna ETH probíhat a že taksakcí bude více. Kanál zjednodušeně funguje tak, že všechny strany uzamknou smluvenou částku ETH do více-podpisového smart kontraktu. Více-podpisový znamená, že potřebuje digitální podpisy od více uživatelů, aby se mohl spustit.

Poté uživatelé mezi sebou provádějí transakce mimo blockchain. Když ukončí transakce, tak vloží seznam transakcí podepsaný všemi stranami do smart kontraktu a tím se uvolní depozit. Ušetří se gas za všechny transakce ETH mezi uživateli, které by se jinak musely jedna po druhé nahrávat do blockchainu.[38]

Kapitola 5

Aplikace PWN

Aplikace PWN byla navržena v roce 2020 během hackatonu Josefem Je. Aplikace je součástí velmi početné rodiny Decentralizovaného Finančnictví (DeFi). Hlavním cílem PWN je přímé propojování (pear-to-pear) uživatelů, kteří chtějí půjčku s věřiteli. Vše probíhá decentralizovaně pomocí smart kontraktů nahraných do Ethereového blockchainu.[4]

Motivace pro použití PWN může být následující. Alice si chce koupit dvě NFT[9] (Non-fungible token), každý za 500$. Ve svém portfoliu má však volných pouze 700$. Proto Alice koupí první NFT, které použije jako zástavu pro půjčku 300$ s tím, že za nějakou dobu 300$ splatí. Věřitelé podle protokolu PWN dávají nabídky půjčky s věřiteli definovaným úrokem. Alice si z těchto nabídek vybere tu, která jí vyhovuje. Řekněme, že si vybrala nabídku Boba s dobou splatnosti 3 měsíce a s úrokem 20$.

Bob jí půjčí 300$ (všechny transakce probíhají decentralizovaně přes PWN smart kontrakty). Alice, která má nyní obnos potřebný pro druhé NFT, si ho koupí. Nyní má 3 měsíce na to si přeuspořádat portfolio, aby splatila 320$ (půjčená částka s úrokem). Pokud toho není schopná, zastavené NFT propadne Bobovi.

5.1 Návrh aplikace

Dlužník má více možností, jak tokeny může zastavit vůči půjčce. Může jít o standardní směnitelné tokeny podle standardu ERC20[10], ale i NFT tokeny ERC721[11] a nakonec hybridní tokeny podle standardu ERC1155[12].[4]

Veškerý proces půjčování a splácení se provádí skrz Deed token, který reprezentuje půjčku. Deed token odpovídá standardu ERC1155 a je vytěžen

[9] NFT je unikátní token. Více na `https://www.theverge.com/22310188/nft-explainer-what-is-blockchain-crypto-art-faq`

[10] `https://docs.openzeppelin.com/contracts/2.x/erc20`

[11] `https://docs.openzeppelin.com/contracts/2.x/erc721`

[12] `https://docs.openzeppelin.com/contracts/3.x/erc1155`

vždy v momentě, kdy je vytvořen požadavek o půjčku dlužníkem a majitel tohoto tokenu je uživatel, který si chce půjčit. Deed token v sobě drží všechny potřebné informace o půjčce. Ve chvíli, kdy uživatel přijme nabídku od věřitele, vlastnictví Deed tokenu přechází na věřitele. Věřitel může s Deed tokenem, tedy s půjčkou, obchodovat.[4]

5.1.1 Vypůjčování si

Jak již bylo řečeno, zástava může být tokenem podle standardů ERC20, ERC-721 nebo ERC1155. Tato zástava přejde do vlastnictví smart kontraktu PWN (uzamkne se tam). Vytvořením žádosti o půjčku se vytvoří Deed token. Žádost o půjčku musí obsahovat, čím bude dlužník věřit a na jak dlouho si chce uživatel půjčit. Požadovaná částka musí být v tokenech podle standartu ERC20 (DAI, Tether...) nebo ERC1155. [4]

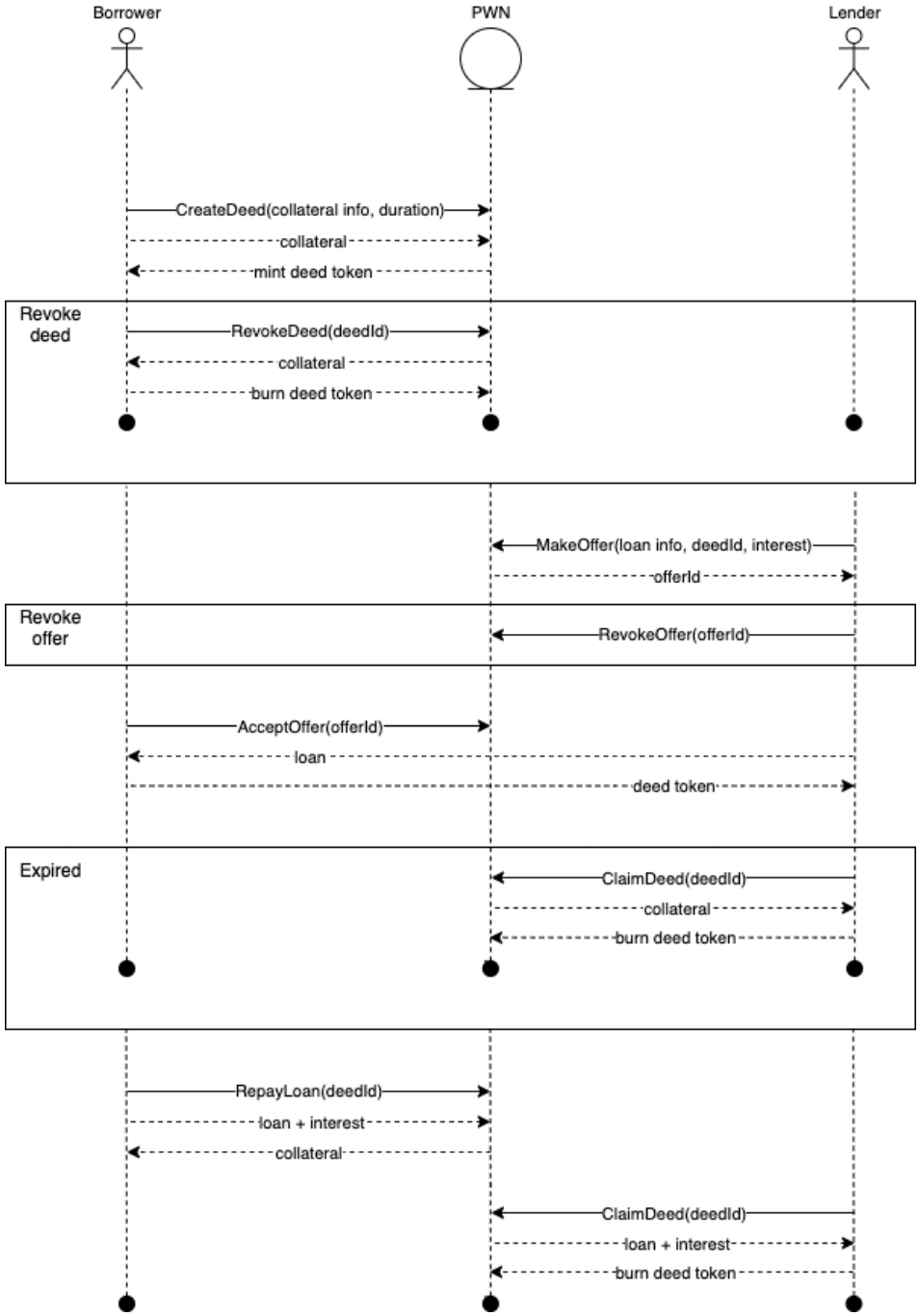
Potencionálnímu dlužníkovi budou chodit na jeho Deed token nabídky. On si může vybrat tu, která je pro něj nejpřijatelnější (podle výše půjčky nebo úroku), a nebo svoji žádost může zrušit. Když nabídku pomocí transakce do PWN smart kontraktu přijme, tak se majitelem Deed tokenu stává ten, kdo danou nabídku učinil. Dlužník má v Deed tokenu definovaný čas na to, aby půjčku splatil i s úrokem. Pokud se tak nestane, věřitel získá zástavu zamknutou v kontraktu.[4]

5.1.2 Půjčování

Potencionální věřitel si vybírá z Deed tokenů, neboli z různých žádostí o půjčku. Věřitel si půjčku vybere a učiní nabídku na token. Nabídka obsahuje, kolik je věřitel podle svého uvážení ochotný za zástavu půjčit a kolik věřitel žádá, aby mu dlužník vrátil (z podstaty věci tato částka bude vyšší, než vypůjčená částka). Věřitel může nabídku na přijmutí půjčky zrušit, pokud ji dlužník již nepřijal.[4]

Když dlužník přijme nabídku věřitele, tak token přejde do vlastnictví věřitele a půjčené peníze jsou poslány uživateli, který si je půjčuje. Věřitel čeká, jestli dlužník peníze splatí (tím že je uzamkne do PWN smart kontraktu), a nebo uběhne doba splatnosti. V obou případech si věřitel pomocí vlastnictví Deed tokenu ze smart kontraktu může něco vyzvednout. Buď splacené peníze, a nebo zástavu. Obrázek 5.1 uvedený v gitu PWN projektu[13], zobrazuje všechny možnosti, které mohou při půjčování nastat.[4]

[13] `https://github.com/PWNFinance/pwn_contracts`



Obrázek 5.1: PWN user flow[4]

5.2 Lokální nasazení

Pro lokální nasazení projektu použijeme kód, který se nachází na Githubu PWN projektu `https://github.com/PWNFinance/pwn_contracts`. Samotné nasazení provedeme pomocí nástroje Brownie[14], který pro lokální nasazení používá simulaci blockchainu nástrojem Ganache[15]. Genache pro nás vytvoří účty, ze kterých můžeme vytvářet transakce a nasazovat smart kontrakty.

Implementace aplikace PWN je v jazyku Solidity a skládá se ze 3 smart kontraktů. První je *PWN.sol*, přes který má chodit většina komunikace při práci s PWN protokolem. Obsahuje funkce pro vytvoření Deed tokenu, zrušení Deed tokenu, vytvoření nabídky pro dlužníka, zrušení nabídky, přijetí nabídky, splacení půjčky a nárokování splátky nebo zástavy majitelem Deed tokenu.

Dalším smart kontraktem je implementace Deed tokeku, která se nachází v souboru *PWNDeed.sol*. Tento token dědí funkce ze standardu ERC1155 z implementace OpenZeppelin[16], který se zabývá standardizací pro smart kontrakty. Dále obsahuje funkce pro správu nebo vytváření Deed tokenu, tyto funkce však může volat pouze PWN kontrakt. Nakonec implementace obsahuje gettery pro získávání hodnot z Deed tokenu.

Posledním smart kontraktem je *PWNVault.sol*. Tento smart kontrakt je uložištěm tokenů, které se během operací do PWN zamykají. Opět většinu funkcí může volat pouze PWN smart kontrakt. To že funkce v *PWNVault.sol* a *PWNDeed.sol*, které mění stav smart kontraktů, může volat jen *PWN.sol* je z bezpečnostního důvodu. Když komunikace uživatele s PWN kontraktem probíhá pouze přes *PWN.sol*, tak je snazší a přehlednější kontrolovat, jestli uživatel na požadované operace má právo.

Pro možnost otestování jsme vytvořili jednoduché testovací tokeny podle standardu ERC20, ERC721 a ERC1155. Jsou k nahlédnutí na přiloženém CD v adresáři *src/contracts/TestTokens*. Příklad testovacího tokenu ERC20 je ukázán ve zdrojovém kódu 5.1.

```
1  // SPDX-License-Identifier: MIT
2  pragma solidity ^0.8.0;
3
4  import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
5
6  contract ERC20MyToken is ERC20 {
7
8      constructor(uint256 initialSupply) ERC20("Fungible Token", "
           FT") {
9          _mint(msg.sender, initialSupply);
10     }
11 }
```

Listing 5.1: Testovací ERC20 token

[14] `https://eth-brownie.readthedocs.io/en/stable/`
[15] `https://docs.nethereum.com/en/latest/ethereum-and-clients/ganache-cli/`
[16] `https://openzeppelin.com/`

Ten, kdo testovací token nahraje na blockchain, může určit, kolik tokenů se má vytěžit a všechny bude vlastnit on.

Nasazení začne tím, že všechny zdrojové .sol kódy zkompilujeme.

```
$ brownie compile
```

Listing 5.2: Kompilace smart kontraktů

Takto nám vznikly byte kódy smart kontraktů, které můžeme nahrát na blockchain. Dále Brownie vytvořilo potřebné ABI, které se používá pro volání funkcí. Můžeme tedy pokračovat a nasadit smart kontrakty na blockchain. Toto budeme dělat z interaktivního prostředí *console* postaveném na Pythonu.

```
$ brownie console
Brownie v1.17.1 - Python development framework for Ethereum

PwnCodereviewProject is the active project.

Launching 'ganache-cli --accounts 10 --hardfork istanbul --
    gasLimit 12000000 --mnemonic brownie --port 8545'...
Brownie environment is ready.
>>>
```

Listing 5.3: Spuštění konzole

Jak je vidět, Brownie vytvořilo simulaci blockchainu pomocí Ganache a vytvořilo pro nás 10 účtů s privátními klíči, které můžeme používat. Účty jsou v seznamu a přistupujeme k nim pomocí klíčového slova *accounts* a indexu identifikující účet. Následně je potřeba importovat všechny zkompilované kontrakty a můžeme je z některého účtu nasadit. Všechny smart kontrakty PWN nasadíme pomocí privátního klíče účtu s indexem 0.

```
>>> from brownie import *
>>> pwn_deed = PWNDeed.deploy("", {'from': accounts[0]})
Transaction sent: 0x6cc8512e3bd9d754a60f4d2dcd29ca906793d51de5a
    ...
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 0
  PWNDeed.constructor confirmed   Block: 1   Gas used: 2718466
      (22.65%)
  PWNDeed deployed at: 0x3194cBDC3dbcd3E11a07892e7bA5c3394048Cc87

>>> pwn_vault = PWNVault.deploy({'from': accounts[0]})
Transaction sent: 0xbe6456d715086cecaa7ddfbc54b175c82cda588e983
...
  PWNVault deployed at: 0
      x602C71e4DAC47a042Ee7f46E0aee17F94A3bA0B6

>>> pwn = PWN.deploy(pwn_deed.address, pwn_vault.address, {'from'
    : accounts[0]})
Transaction sent: 0
    x35c92b09ad238fd2ce34137616044d9978043f062fbb0cf86def6d09...
...
  PWN deployed at: 0xE7eD6747FaC5360f88a2EFC03E00d25789F69291
```

Listing 5.4: Nasazení smart kontraktů na blockchain

PWN kontrakty máme nasazené. Nyní musíme nastavit adresu *PWN.sol* kontraktu kontraktům *PWNDeed* a *PWNVault*. Toto nastavení je z toho důvodu, aby PWNDeed a PWNVault mohly kontrolovat, že jejich funkce volá pouze kontrakt PWN a tím se zamezilo, aby uživatelé mohly volat funkce PWNDeed a PWNVault přímo a mohli nekontrolovaně měnit data.

```
>>> pwn_deed.setPWN(pwn.address, {'from': accounts[0]})
...

>>> pwn_vault.setPWN(pwn.address, {'from': accounts[0]})
...
```

Listing 5.5: Nastavení adresy PWN kontraktu

Nyní nasadíme testovací tokeny. Token podle standardu ERC20 bude reprezentovat platidlo, které si chce dlužník vypůjčit. ERC721 bude NFT, které dlužník zastavuje. Účet 1 nasadí a vytěží NFT a účet s indexem 2 nasadí a vytěží platící token a část tokenů pošle účtu 1, aby později měl čím splácet.

```
>>> erc20 = ERC20MyToken.deploy(1000, {'from': accounts[2]})
Transaction sent: 0
    xa049744e8875dda89e6279b873055be94e14cd4074debbfea0cfc...
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 0
  ERC20MyToken.constructor confirmed   Block: 6   Gas used:
      637074 (5.31%)
  ERC20MyToken deployed at: 0
      x1596Ff8ED308a83897a731F3C1e814B19E11D68c

>>> erc20.transfer(accounts[1], 200, {'from': accounts[2]})
...

>>> erc20.balanceOf(accounts[1])
200
>>> erc20.balanceOf(accounts[2])
800

>>> erc721 = ERC721MyToken.deploy("Valuable token", {'from':
    accounts[1]})
...
  ERC721MyToken deployed at: 0
      xe7CB1c67752cBb975a56815Af242ce2Ce63d3113

>>> erc721.balanceOf(accounts[1])
1
```

Listing 5.6: Rozdělení testovacích tokenů

Pro shrnutí, účet 0 simuluje majitele PWN a nahrál kontrakty PWN, PWNDeed a PWNVault. Účet 1 vlastní jedno NFT jménem „Valuable token“ a 200 platících tokenů. Nakonec účet 2 reprezentuje věřitele a vlastní 800 platících tokenů.

Účet 1 vytvoří Deed token s žádostí o půjčku tím, že volá funkci *createDeed*, která je součástí PWN kontraktu. Funkce bude chtít zamknout do PWNVault

token, kterým účet 1 ručí. Proto účet musí povolit, aby PWNVault mohl tento token přesunout k sobě. Implementace funkcí jsou k nalezení ve výše zmiňovaném gitu nebo v přiloženém CD.

```
>>> erc721.approve(pwn_vault.address, 0, {'from': accounts[1]})
>>> pwn.createDeed(erc721.address, 1, 3600, 0, 1, {'from':
    accounts[1]})
...
# ucet 1 vlastni 1 Deed token
>>> pwn_deed.balanceOf(accounts[1], 1)
1
# vault vlastni 1 NFT token
>>> erc721.balanceOf(pwn_vault.address)
1

>>> pwn.revokeDeed(1, {'from': accounts[1]})
...
>>> erc721.balanceOf(accounts[1])
1
>>> erc721.balanceOf(pwn_vault.address)
0
```

Listing 5.7: Vytvoření a zrušení Deed tokenu

Vytvořili jsme žádost o půjčku s dobou splatnosti 3600 vteřin (od přijetí nabídky věřitelem), kde ručíme tokenem podle standardu ERC721. Účet 1 vlastnil Deed token s hodnotou ID 1. Následně jsme žádost o půjčku zrušili a zastavený token se opět stal majetkem účtu 1.

Nyní již nově vytvořený Deed token (s ID 2) rušit nebudeme, ale vytvoříme na něj nabídku půjčky 100 tokenů s vrácením 120 tokenů účtem 2, kterou účet 1 přijme. Nesmíme zapomenou nastavit, že účet 2 povoluje transakci 100 tokenů ze svého účtu na adresu PWNVault, který v sobě uchovává tokeny v rámci PWN. Aby přijmutí nabídky bylo možné, musí účet 1 povolit manipulaci se svým Deed tokenem. Ten po přijetí nabídky změní majitele, majitelem se stane účet 2. Povolení k manipulaci musí být uděleno PWNVault, které pomocí funkce pullProxy zprostředkovává transakce.

K přijmutí nabídky musíme znát ID nabídky. To můžeme získat podle logu z transakce pro vytvoření nabídky. Tento log se vytvořil pomocí eventů. Další možnost je pomocí getteru *getOffers*.

```
    pwn.makeOffer(erc20.address, 100, 2, 120, {'from': accounts
        [2]})
...
Events In This Transaction
--------------------------
 |__ PWNDeed (0x3194cBDC3dbcd3E11a07892e7bA5c3394048Cc87)
    |__ OfferMade
        |__ assetAddress: 0
            x1596Ff8ED308a83897a731F3C1e814B19E11D68c
        |__ amount: 100
        |__ lender: 0x0063046686E46Dc6F15918b61AE2B121458534a5
        |__ toBePaid: 120
```

```
        |__ did: 2
        |__ offer: 0x4bc7c78f9ecfb80d3468b0fcf9f9c0dd594088...
        # zde je ID nabidky

>>> erc20.approve(pwn_vault.address, 100, {'from': accounts[2]})
...

>>> pwn_deed.setApprovalForAll(pwn_vault.address, 1, {'from':
    accounts[1]})
...

>>> tx = pwn.acceptOffer(0x4bc7c78f9ecfb80d3468b0fcf9f9c0dd59...,
     {'from': accounts[1]})
...

>>> pwn_deed.balanceOf(accounts[2], 2)
1
>>> erc20.balanceOf(accounts[1])
300
```

Listing 5.8: Přijmutí nabídky na půjčku

Přijetí nabídky bylo úspěšné a účet 1 má nyní o 100 platících tokenů vyšší zůstatek a účet 2 vlastní Deed token.

V tuto chvíli účet 1 splatí půjčku a účet 2 si vyzvedne vrácené tokeny i s úrokem, který činí 20 tokenů. Účtu 1 se vrátí NFT token, kterým ručil.

```
>>> erc20.approve(pwn_vault.address, 120, {'from':accounts[1]})
...

>>> pwn.repayLoan(2, {'from': accounts[1]})
# 2 je ID Deed tokenu
...

>>> erc20.balanceOf(pwn_vault.address)
120
# vidime, ze do pwn_vault se zamklo 120 vracenych
# tokenu a musi si je vyzvednout pouze majitel
# prislusneho Deed tokenu

>>> pwn.claimDeed(2, {'from': accounts[2]})
...

>>> erc20.balanceOf(accounts[1])
180
>>> erc20.balanceOf(accounts[2])
820
>>> erc721.balanceOf(accounts[1])
1
```

Listing 5.9: Splacení půjčky

Nyní zkusíme možnost, že dlužník nestihne včas splatit půjčku. NFT token by tedy měl propadnout věřiteli a měl by si ho moci ze smart kontraktu PWNVault vyzvednout.

```
# vynechavame prikazy pro povoleni transakci
>>> pwn.createDeed(erc721, 1, 60, 0, 1, {'from': accounts[1]})
# pouze 60 vterin pro splaceni pujcky
...

>>> pwn.makeOffer(erc20.address, 100, 3, 120, {'from': accounts
    [2]})
...

>>> pwn.acceptOffer(0x72b624bb76c9de4daa490994dc70c5..., {'from':
     accounts[1]}
# pockame vice nez 60 vterin

>>> pwn.repayLoan(120, {'from': accounts[1]}) ERROR!
# nelze splatit po casu splatnosti

>>> pwn.claimDeed(3, {'from': accounts[2]})

>>> erc721.balanceOf(accounts[2])
1
# ucet 2 ziskal zastavu v podobe NFT
```

Listing 5.10: Expirování Deed tokenu

Skutečně se tak stalo. Tímto modelovým příkladem jsme ukázali, že aplikace funguje podle očekávání. Obecně je ale nepraktické používat pouze *Brownie* konzoly, protože není možné vytvářet skripty, pokud chceme operace často opakovat.

Brownie umožňuje všechny transakce psát do python skriptu a ten poté spustit. Všechny transakce se vykonají na specifikovaném blockchainu. V našem případě bude to opět simulace blockchainu pomocí Ganache.

Skript pro nasazení kontraktů, vykonání zastavení NFT pro půjčku ERC20 tokenů, splacení tokenů s úrokem a vyzvednutí těchto tokenů věřitelem je součástí přiloženého CD jako soubor *src/scripts/deploy_pwn.py*. Skript je možné spustit příkazem *brownie run scripts/deploy_pwn.py*.

5.3 Unit testování

V této podkapitole se budeme věnovat unit testům, které jsou vytvořeny pro kontrolu, že aplikace funguje tak za všech okolností, jak je zamýšleno. K testování funkcí použijeme aplikaci Brownie, které má testování postavené na pytestech. Pomocí parametru *coverage* je možné určit, kolik procent kódu funkce bylo při testech spuštěno. Pokrytí funkce se spočítá ze spuštěného byte kódu při testování. Cílem je, aby každá testovaná funkce měla pokrytí alespoň 90% testy, které napíšeme.

PWN aplikace je tvořena ze 3 smart kontraktů. Hlavního *PWN.sol*, který volá funkce zbylých dvou kontraktů *PWNDeed.sol* a *PWNVault.sol*. Kontrakty *PWNDeed.sol* a *PWNVault.sol* mají v sobě nastavenou kontrolu, že funkce,

které mění vnitřní stavy těchto kontraktů, mohou být zavolány pouze *PWN.sol* kontraktem. Tato kontrola se provádí tím, že se kontraktům *PWNDeed.sol* a *PWNVault.sol* nastaví adresa *PWN.sol* a při každém zavolání funkce se zkontroluje pomocí této adresy, jestli je volající skutečně kontrakt *PWN.sol*. Gettery pro čtení hodnot vnitřního stavu kontraktu může volat kdokoliv, to není omezeno pouze pro *PWN.sol*.

Deed token si interně drží 5 stavů, které říkají, jaké operace se s ním mohou provádět. To dobře zabezpečuje kontrakt před nežádoucím manipulováním. Následující seznam uvádí stavy a jaké funkce se mohou v daném stavu volat.

- Stav 0 znamená spálený token. Do tohoto stavu se token dostane tehdy, když byl Deed token zrušen ještě před přijmutím nabídky nebo proběhl celý proces a věřitel si vybral odměnu. Ve stavu 0 nelze s tokenem provádět žádné operace.
- Stav 1 je otevřený token. Token může být zrušen nebo může přijmout nabídku. Žádná jiná operace není s tokenem v tomto stavu možná.
- Běžící token je ve stavu 2. To znamená, že dlužník akceptoval nabídku a nyní je možné zavolat funkci pro splacení půjčky.
- Stav 3 je již splacený token. Tento stav nastane při splacení a ve stavu 3 je pouze možné vybírat odměnu věřitelem.
- Stav 4 reprezentuje přesažení doby pro splacení půjčky. Ve stavu 4 je pouze možné, aby si věřitel vybral zástavu, kterou dlužník ručil.

V testování se zaměříme na funkce *PWN.sol* kontraktu, protože ty jediné dokáží měnit vnitřní stavy kontraktů a provádí logiku za touto DeFi aplikací.

PWN.sol kontrakt je tvořen konstruktorem, který má parametry adresy kontraktů *PWNDeed.sol* a *PWNVault.sol*, které se uloží do členských proměnných *PWN.sol*. Toto je nezbytné, aby *PWN.sol* kontrakt věděl, na jaké adresy posílat transakce.

```
constructor(address _PWND, address _PWNV) Ownable() {
    deed = PWNDeed(_PWND);
    vault = PWNVault(_PWNV);
}
```

První testovanou funkcí je funkce *createDeed*, která vytvoří Deed token podle parametrů a zamkne token, kterým uživatel ručí, do *PWNVault.sol* kontraktu. Vytváření respektive zamknutí tokenu se provede voláním funkcí *PWNDeed.create()* respektive *PWNVault.push()*. Funkce *PWNDeed.create()* nastaví stav Deed tokenu na hodnotu 1.

```
function createDeed(
    address _assetAddress,
    MultiToken.Category _assetCategory,
    uint32 _duration,
```

```
    uint256 _assetId,
    uint256 _assetAmount
) external returns (uint256)
```

Následující funkce *revoke*, kterou testujeme, zruší Deed token, ale tuto operaci může vykonat pouze vlastníkem Deed tokenu a pouze tehdy, jestli je token ve stavu 1.

```
function revokeDeed(uint256 _did) external
```

Funkce *makeOffer* udělá nabídku tokenu podle parametrů. Například parametr *_did* odkazuje na unikátní identifikátor Deed tokenu, kterému se má nabídka vytvořit. Nabídka se vytvoří pouze v případě, že Deed token je ve stavu 1.

```
function makeOffer(
    address _assetAddress,
    uint256 _assetAmount,
    uint256 _did,
    uint256 _toBePaid
) external returns (bytes32)
function revokeDeed(uint256 _did) external
```

Naopak funkce *revokeOffer* zruší nabídku podle identifikátoru nabídky, ale jen v tom případě, když ten, kdo chce nabídku zrušit, ji vytvořil a Deed token je ve stavu 1.

```
function revokeOffer(bytes32 _offer) external
```

Funkcí *acceptOffer* přijmeme nabídku podle jejího identifikátoru. Transakce volající funkci *acceptOffer* musí splnit, že transakci podepsal vlastník daného Deed tokenu a že token je ve stavu 1. Přijmutím nabídky nastavíme status Deed tokenu na 2.

```
function acceptOffer(bytes32 _offer) external returns (bool)
```

Další testovanou funkcí je funkce *repayLoan*. Pokud je Deed token ve stavu 2 a jsme jeho majitelé, tak touto funkcí splatíme půjčku a ta se zamkne do *PWNVault.sol* kontraktu. Deed token bude po úspěšném zavoláním funkce ve stavu 3.

```
function repayLoan(uint256 _did) external returns (bool)
```

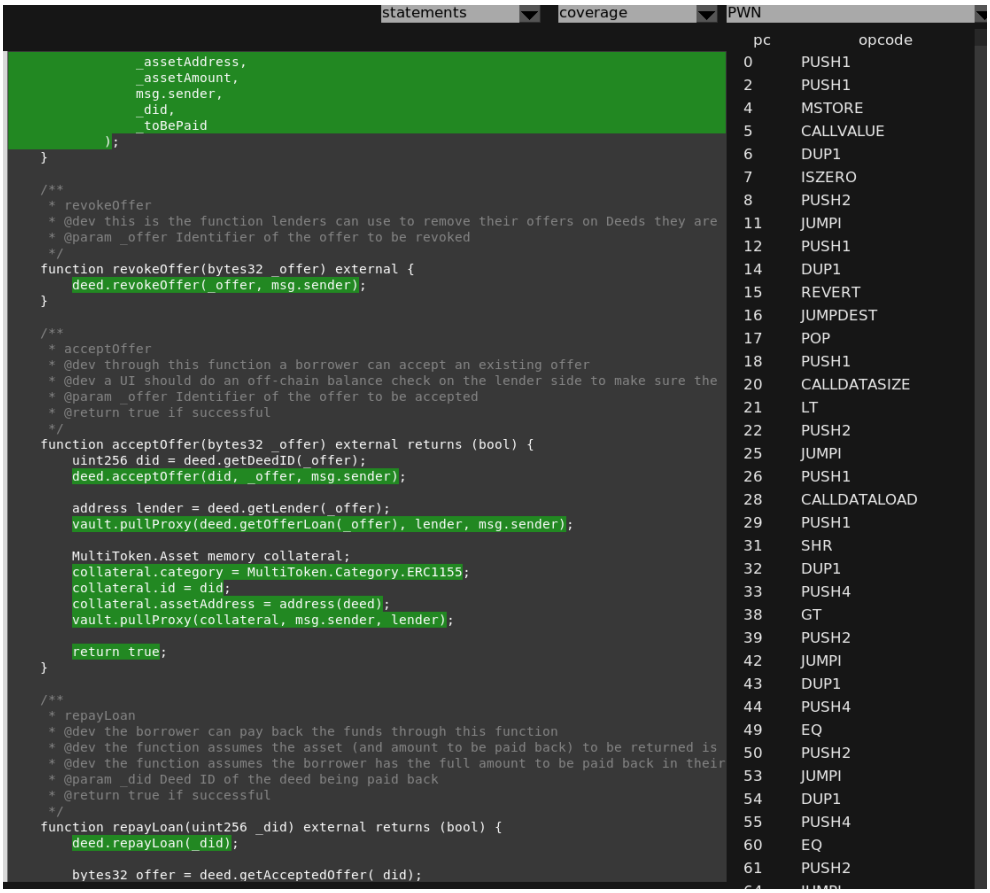
Poslední funkce se jmenuje *claimDeed*. Pokud ten, kdo tuto funkci volá, je současný majitel Deed tokenu a token je ve stavu 3, tak si tímto voláním vyzvedne odměnu zamčenou v *PWNVault.sol* kontraktu.

```
function claimDeed(uint256 _did) external returns (bool)
```

Kontrakty importují další již implementované kontrakty a poté z těchto kontraktů dědí. Například Deed token dědí z ERC1155 implementace, kterou vytvořila skupina OpenZeppelin, a tím podědí funkce na manipulaci s tokenem podle standardu ERC1155. OpenZeppelin je běžně používaná knihovna, a proto její funkce považujeme za bezpečné a nebudeme je nijak testovat.

Testy pro všechny výše zmíněné funkce jsou napsané v pythonu v souboru *tests/test_pwn.py* a jsou napsané tak, že mají pokrytí 100% ve všech funkcích, které mění vnitřní stav v kontraktech *PWN.sol*, *PWNDeed.sol* a *PWNVault.sol*.

Testy spustím pomocí příkazu *brownie test coverage* a tak se spočítá pokrytí veškerého kódu ve všech zkompilovaných smart kontraktech. Příkaz mimo jiné vypíše pokrytí testy kódu jednotlivých funkcí, ale jen pro funkce, které mají v sobě rozvětvení (například if nebo require). Proto je lepší zapnout Brownie GUI příkazem *brownie gui*, nastavit požadovaný kontrakt, vybrat položku coverage a statements pro běžný kód nebo branches pro rozvětvení. V GUI se barevně označí kód, který byl při posledních testech spouštěn, jak je vidět na obrázku 5.2. Jak je možné si zkontrolovat, všechen kód byl v kontraktech *PWN.sol*, *PWNDeed.sol* a *PWNVault.sol* spuštěn a testy, které jsme napsali, pokryly 100% byte kódu funkcí smart kontraktů *PWN.sol*, *PWNDeed.sol* a *PWNVault.sol*.



Obrázek 5.2: Brownie GUI pro pokrytí testy

Všechny spuštěné testy prošly až na ten poslední. Ten testuje půjčení tokenu ERC1155 dlužníkovi, to by podle white paperu [4] mělo být možné. Vzhledem k tomu, že však test této funkcionality neprošel, tak v tomto případě aplikace nefunguje, jak je předpokládáno. Tuto chybu rozvedeme v podkapitole 5.6.2, věnující se manuálnímu code review.

5.4 Slither

Slither[17] je nástroj pro statickou analýzu kódu psaném v Solidity. Spustí baterii testů na známé zranitelnosti a v případě, že nalezne podezření na zranitelnost, tak ji vypíše v reportu.

Příkazem *slither src/slither/pwn_contracts/contracts/PWN.sol* spustíme analýzu a Slither od sebe barevně odliší závažnost zranitelnosti. Červně je závažná zranitelnost, žlutě středně závažná a zeleně málo závažná zranitelnost. Výsledky analýzy pro importované knihovny OpenZeppelinu nebudeme zpracovávat, protože jak jsme již uvedli, považujeme implementaci této knihovny za bezpečnou. Výsledek analýzy je uložen v souboru *slither/results.txt*.

5.4.1 Závažné zranitelnosti

Závažná zranitelnost se podle Slitheru nachází v knihovně MultiToken od PWNFinance[18]. Tato knihovna je od stejných autorů jako PWN, proto se touto zranitelností budeme zabývat.

Slither říká, že knihovna MultiToken.sol na řádcích 45[19] respektive 73[20] nekontroluje návratovou adresu z funkce *transfer* respektive *transferFrom*. Tyto funkce se používají na přenos tokenů od jednoho uživatele k druhému a jsou poděděné z implementace ERC20 OpenZeppelinu.

Některé implementace tokenů v případě neúspěchu transakce vrací false z funkce a nerevertují transakci, ale v případě OpenZeppelinu tomu tak není. OpenZeppelinu vždy operaci kvůli chybě revertuje a nevrací false. Nejedná se tedy o skutečnou zranitelnost, ale návratové hodnoty by se měly obecně kontrolovat. Na druhou stanu, kontrola podmínky stojí gas.

5.4.2 Středně závažné zranitelnosti

Jako první středně závažnou zranitelnost označil Slither kontrolu stavu Deed tokenu v PWN jednotlivých funkcích pro manipulaci s tímto tokenem. Slither

[17] `https://github.com/crytic/slither`

[18] `https://github.com/PWNFinance/MultiToken`

[19] `https://github.com/PWNFinance/MultiToken/blob/e5f38b5606f20c56c5ac4097ef7a1aab80e67fe4/contracts/MultiToken.sol#L45`

[20] `https://github.com/PWNFinance/MultiToken/blob/e5f38b5606f20c56c5ac4097ef7a1aab80e67fe4/contracts/MultiToken.sol#L73`

říká, že podmínky jsou příliš striktní[21] a bojí se, že může nastat stav, kdy nebudou nikdy splněny a nebude s Deed tokenem vůbec možné manipulovat. V tomto případě je striktní podmínka „==“ nutná pro určení, v jakém stavu se Deed token nachází a jaké funkce se pro něj mohou vykonávat. Proto toto nepovažujeme za zranitelnost.

Další Slitherem označená středně závažná zranitelnost je, že v kontraktu *PWNDeed.sol* ve funkci *create* je nejprve použito externí volání funkce *_mint* a až poté se nastaví hodnota statusu Deed tokenu. Je zde podezření na re-entrancy[22], ale tato slabina by mohla vzniknout pouze v případě, že bychom volali nějakou funkci v nedůvěryhodném kontraktu. *_mint* je poděděná z implementace ERC1155 tokenu od OpenZeppelinu a tuto implementaci považujeme za bezpečnou. Proto ani toto nepovažujeme za reálnou slabinu.

Následující zranitelností má být, že lokální proměnná *collateral* není v kontaktu *PWN.sol* ve funkci *acceptOffer* nikde inicializovaná. Slither nedokázal rozpoznat, že *collateral* je struktura, která je na následujících řádcích inicializovaná.

Poslední středně závažnou zranitelností podle Slitheru je, že se v kontaktu *PWN.sol* nikde nekontroluje návratová hodnota z funkcí *PWNVault.pull* a *PWNVault.push*. To je pravda, ale funkce pull push nikdy nevrací false a v případě chyby revertují transakci. Proto toto nepovažujeme za zranitelnost, kterou by mohl útočník využít, ale návratová hodnota by se měla kontrolovat nebo by ji funkce neměla vracet.

5.4.3 Málo závažné zranitelnosti

První málo závažnou potencionální zranitelností je, že *PWNDeed.sol* používá jména *_uri* a *_owner* jako parametry funkcí, ale tato jména již používají jako členské proměnné kontrakty, ze kterých *PWNDeed.pull* podědil. V některých případech může dojít k záměně proměnných, ale v této implementaci je všechno v pořádku.

Další možná zranitelnost je, že se ve funkci *PWNDeed.setPWN* nekontroluje, jestli je nastavená nulová adresa. Tuto funkci může volat pouze majitel *PWN.sol* kontraktu. Bez správného nastavení PWN adresy nejde volat žádná funkce, která mění vnitřní stav kontraktu, proto tuto kontrolu nedělat nepředstavuje žádné riziko a ušetří se gas.

Následně si Slither stěžuje, že se emitují eventy až po zavolání externí funkce. Mohlo by tedy opět docházet k re-entrancy. Zde platí stejný argument, který jsme napsali v podkapitole 5.4.2 ke stejné potencionální zranitelnosti.

Následující poznámka Slitheru odkazuje na to, že se okamžik, kdy dlužník nestihl splatit dluh, odvozuje od časové známky, kdy byl blok vytěžen. Toto nemusí být bezpečné, protože tento okamžik nastavuje uživatel, který blok

[21] `https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities`

[22] `https://github.com/crytic/not-so-smart-contracts/tree/master/reentrancy`

vytěžil, a může s tímto časem manipulovat. Tuto možnou zranitelností se budeme více zabývat v podkapitole 5.6.

Zbylé nálezy, které Slither udělal, se týkají spíše konvence pojmenování proměnných, přílišná podobnost při pojmenování proměnných a podobně. Tímto se zde nebudeme dále zabývat, protože nic z toho nezpůsobuje zranitelnost aplikace. Tyto nálezy jsou však vypsané v souboru *src/slither/results.txt*, který se nachází na přiloženém CD.

5.5 Fuzzy testy

Fuzzy testování má za úkol odhalit chyby v programech tak, že zkouší volat funkce s různou hodnotou parametrů a najít takovou kombinaci parametrů, aby nastal neočekávaný stav. Hodnoty těchto parametrů mohou být zcela náhodné, nebo se může cílit na krajní meze. Záleží na tom, jak jsou testy nastaveny. Další způsob, kterým fuzzy testy hledají chyby, je, že spouštějí funkce v různém pořadí, dávají jim různé parametry a opět se hledá nečekané chovaní programu. Fuzzy testy se nejvíce hodí na funkce, které provádějí matematické výpočty a jím podobné.

Fuzzy testy pro PWN aplikaci nepřinesou nic nového oproti unit testům popsaných v podkapitole 5.3. Je to z toho důvodu, že vstupy do jednotlivých funkcí jsou ID jednotlivých tokenů nebo nabídek. Toto nelze testovat náhodnými daty, protože ID tokenu buď existuje a poté s tímto tokenem lze pracovat, a nebo ne. Pokud ID tokenu neexistuje, PWN aplikace správně vyhodí výjimku, ale to jsme již otestovali v unit testech. Proto by v tomto případě fuzzy testy byly redundantní. V případě funkcí *createDeed* a *makeOffer* jsou i jiné parametry než jen ID tokenů. Například délka půjčky nebo půjčená suma, ale tyto hodnoty se pouze zkopírují do tokenu. Všechny základní operace, které se s těmito hodnotami provádějí, jsou pokryté v unit testech.

Náhodné volání funkcí a pozorování, jestli se změnila nečekaně nějaká vnitřní hodnota v PWN aplikaci, není jednoduché. Protože Deed token je vždy v nějakém stavu, který umožňuje volání jen malé množině funkcí, které token přesunou do jiného stavu, jak je popsáno v 5.3. Tuto funkcionalitu jsme pečlivě otestovali v unit testech, a proto by fuzzy testy opět nepřinesly nic nového.

Jak bylo řečeno výše, fuzzy testování se hodí na funkce, které provádějí výpočty, které závisejí na parametrech, se kterými je funkce volána. V PWN aplikaci jsou funkce přímočaré a mají jasně dané pořadí, ve kterém se mohou vykonávat. Protože unit testy jsou v tomto případě dostatečné, rozhodli jsme se fuzzy testování nedělat

5.6 Manuální code review

Nyní je čas přistoupit k manuální analýze kódu. Zde se zabýváme bezpečnostní stránkou kódu, celkovým návrhem aplikace a jak se implementace aplikace držela specifikace ve white paperu[4].

5.6.1 Timestamp v bloku

Pro určení, jestli je Deed token ve stavu 4 (dlužník nestihl včas splatit dluh) se používá časová známka umístěna v bloku.

```
function getDeedStatus(uint256 _did) public view returns (uint8)
    {
    if (
        deeds[_did].expiration > 0 &&
        deeds[_did].expiration < block.timestamp &&
        deeds[_did].status != 3
    ) {
        return 4;
    } else {
        return deeds[_did].status;
    }
}
```

Hodnotu časové známky pro blok určuje miner, který blok vytěžil. Musí splnit podmínku, že její hodnota musí být větší, než je časová známka předchozího bloku a není o více než 900 vteřin starší[23], než je skutečný čas, kdy blok ostatní mineři schvalují.

Pokud by věřitel vlastnil node, který by vytěžil blok v době, kdy by byla doba splatnosti půjčky méně než čtvrt hodiny, mohl by věřitel umístit svou transakci pro výběr zástavy do tohoto bloku a nastavit časovou známku o 900 vteřin větší než je současný čas. To by znamenalo, že kontrola statusu by vzala tento posunutý stav a chybně by označila Deed token za prošlý. Jedná se o velice nepravděpodobný a teoretický útok. Muselo by se sejít velké množství okolností, aby nastal, ale stále je možný. Útočník by musel dokázat vytěžit blok čtvrt hodiny před expirací půjčky. Při výpočetní složitosti POW, by musel mít útočník velkou výpočetní sílu nebo štěstí. Také půjčky jsou zpravidla dlouhodobé, takže čtvrt hodina ve většině případů nehraje roli. Nicméně je potřeba si být této možnosti vědom. Použití časové známky pro něco tak důležitého, jako je určení času propadnutí zástavy dlužníka věřiteli, nepovažujeme za vhodné.

5.6.2 Nesplnění specifikace

PWN aplikace má na svých stránkách uvedený white paper[4], kde je popsáno, jak aplikace funguje a co by měla umět. V některých bodech se ale implementace od white paperu liší.

[23] https://github.com/ethereum/wiki/blob/c02254611f218f43cbb07517ca8e5d00fd6d6d75/Block-Protocol-2.0.md

První nesplnění specifikace je, že se ve white paperu píše: „The borrowing party sets an expiration date at creation. A recommended duration of a deed is 1 - 3 months. The duration won't be adjustable based on when an offer is accepted creating an incentive to accept offers rather sooner than later.“[4]. To není pravda v tom, že strana, která si půjčuje, nastavuje pouze dobu v sekundách, za kterou se má token splatit. Dlužník nenastavuje datum, ale nastavuje dobu. Další nekonzistence je, že se datum splatnosti nastaví součtem chvíle, kdy byla nabídka akceptována a dobou definovanou dlužníkem. Datum splatnosti se mění podle chvíle, kdy byla nabídka přijata a ne tak, jak to tvrdí white paper.

```
  function acceptOffer(
       uint256 _did,
       bytes32 _offer,
       address _owner
   ) external onlyPWN {
     ...
     deed.expiration = uint40(block.timestamp) + deed.duration;
     deed.acceptedOffer = _offer;
     delete deed.pendingOffers;
     deed.status = 2;
     ...
}
```

Druhé nesplnění specifikace je, že věřitel může token půjčovat pouze tokeny podle standardu ERC20, nikoliv podle standardu ERC1155. Ve white paperu se ale píše: „To make an offer a lender has to specify the following: An asset - address a ERC20 or ERC1155 compliant fungible token“. Z toho jasně plyne, že PWN má umět půjčovat i ERC1155 tokeny. Když jsme tuto funkcionalitu testovali v podkapitole 5.3, tak toto byl jediný test, který neprošel.

Důvodem je, že PWN aplikace nemá dotaženou implementaci pro půjčování tohoto tokenu. Funkce *makeOffer* má parametry pouze adresu tokenu, sumu tokenu, kterou věřitel půjčí, ID Deed tokenu a kolik tokenů požaduje věřitel splatit. Tyto parametry jsou dostatečné pro ERC20 token, kde jsou všechny tokeny stejné a nerozlišují se přes ID. ERC1155 však funguje jinak, jedná se o unikátní token, který je identifikovatelný právě přes dané ID. Proto, když dlužník chce přijmout nabídku na půjčení ERC1155 tokenu, tak PWN nemá informaci o ID ERC1155 tokenu a neví, který poslat dlužníkovi. Transakce se neprovede a skončí chybou.

```
function makeOffer(
    address _assetAddress,
    uint256 _assetAmount,
    address _lender,
    uint256 _did,
    uint256 _toBePaid
) external onlyPWN returns (bytes32) {
    ...
    Offer storage offer = offers[hash];
    offer.loan.assetAddress = _assetAddress;
```

```
    offer.loan.amount = _assetAmount;
    offer.toBePaid = _toBePaid;
    offer.lender = _lender;
    offer.did = _did;
```

Navrhli jsme opravení tohoto nedostatku. Stačí jako parametr funkce *makeOffer* v souboru *PWN.sol* a *PWNDeed.sol* přidat ID ERC1155 a jaký typ tokenu se půjčuje, aby PWN vědělo, jakou funkci pro přenos použít. Poté je nutné tyto parametry uložit do struktury *loan* tak, jak jsme zobrazili v následujícím kódu. Po této opravě PWN aplikace umí půjčovat i tokeny podle standardu ERC1155.

```
function makeOffer(
    address _assetAddress,
    uint256 _assetAmount,
    uint256 _assetID,
    MultiToken.Category _category,
    address _lender,
    uint256 _did,
    uint256 _toBePaid
) external onlyPWN returns (bytes32) {
    ...
    Offer storage offer = offers[hash];
    offer.loan.assetAddress = _assetAddress;
    offer.loan.amount = _assetAmount;
    offer.loan.category = _category;
    offer.loan.id = _assetID;
    offer.toBePaid = _toBePaid;
    offer.lender = _lender;
    offer.did = _did;
```

Nedodržení white paperu v implementaci pouvažujeme za velký problém. Při použití uživatel očekává, že aplikace bude fungovat, jak je ve white paperu napsáno, a nebude pokaždé studovat kód. U aplikací, které pracují s finančními prostředky, je nedodržení specifikace závažné a výrazně podkopá důvěru v aplikaci. Já osobně bych po takovémto zjištění tuto aplikaci pro finanční operace nepoužil.

5.6.3 Návrh aplikace

Samotný návrh aplikace je nešikovný. Na vykonání celého cyklu od vytvoření Deed tokenu do jeho splacení je zapotřebí minimálně 5 transakcí do PWN kontraktu. Do toho nepočítáme transakce pro povolení přenosu tokenů do PWNVault, ale tomu se při žádném návrhu nejde vyhnout. Věřitel musí provést nabídku a poté si vybrat zisk. Podle simulace pomocí Ganache spálí věřitel těmito operacemi přibližně 251 500 gasu. Dlužník musí vytvořit Deed token, přijmout nabídku a splatit ji. To ho bude stát okolo 410 000 gasu.

Následně určeme base fee jako 50 gwei a spropitné 1,5 gwei, tyto hodnoty jsme určili podle průměrných historických hodnot za poslední týden [24].

[24] https://etherscan.io/gastracker#historicaldata

Poté již je snadné spočítat, jaký bude poplatek podle již definované rovnice v podkapitole 3.6.

$$fee = gasUnits * (baseFee + tip)$$

$$veritelTransactionFee = 251\ 500 * (50 + 1.5) = 12\ 952\ 250\ gwie$$

$$dluznikTransactionFee = 410\ 000 * (50 + 1.5) = 21\ 125\ 454\ gwie$$

Gwei je 10^{-9} ETH a při ceně 3 000$ za ETH bude věřitel za všechny operace platit 39$ a dlužník 63$. Tyto ceny jsou ještě spíše optimistické.

V každém případě platit takového částky pouze na poplatcích se nám zdá být příliš vysoká cena za decentralizované půjčky. Lepší návrh aplikace, kde by nebylo potřeba tolik transakcí, by mohl ušetřit uživatelům hodně peněz.

Další nevýhodou současného návrhu, kterou spatřujeme, je, že není možné kontrolovat, jestli má při vytvoření nabídky potencionální věřitel dostatečný zůstatek. Nečestný věřitel by mohl dávat výhodné nabídky, ale neměl by dostatečný zůstatek nebo by nedal povolení k transakci. Kdyby chtěl dlužník takovouto nabídku přijmout, transakce by skončila s chybou. Dlužník by ale stejně za tuto transakci musel spotřebovaný gas zaplatit, konkrétně 102 500 gasu. To vychází podle výše definovaných konstant na 16$. Tímto způsobem by mohli nečestní věřitelé zbavovat dlužníky peněz.

Pro obě výše popsané slabiny navrhujeme jako řešení přenést část operací mimo blockchain, a ten používat k uložení již schválených dohod a k uzamčení tokenů.

Fungovalo by to tak, že by se na klasickém webovém serveru ukazovaly různé půjčky a čím by se za půjčku ručilo. Potencionální věřitelé by mohli na půjčky dělat nabídky a z těchto nabídek by si dlužníci vybírali. Server by kontroloval, že jak dlužník tak věřitel má dostatečný zůstatek pro uzavření dohody. Toto ověření nestojí žádný ETH, protože tyto informace jsou uložené blockchainu, a stačí si tedy projít databázi plného nodu, což je veliká výhoda.

Pokud jsou zůstatky v pořádku, může dlužník nabídku přijmout a udělá se první transakce do blockchainu. Ta by měla v sobě uložené všechny potřebné informace o půjčce a obsahovala by podpis obou uživatelů, že s touto transakcí souhlasí. Vytěžil by se Deed token, který by se dal věřiteli, zástava by se zamkla do kontraktu a dlužník by získal půjčenou sumu.

Poté by zbývala pouze jediná transakce, a to v momentě, kdy by dlužník chtěl Deed token splatit. Poslal by dlužnou částku do PWN a to by ji přeposlalo věřiteli. Následně by PWN vrátilo zastavený token a Deed token by se spálil.

V případě, že by Deed token expiroval, věřitel by si vyžádal zastavený token, PWN by mu ho vydalo a spálilo by daný token. V každém případě by se již provedla pouze jedna transakce do blockchainu.

V součtu by se do blockchainu provedly dvě transakce místo pěti a vyřešily by se tím oba výše popsané problémy. Další velkou výhodou by bylo, že by PWN kontrakt nepoužíval tolik paměti v Blockchainu, protože v současném návrhu se všechny návrhy půjček ukládají pravě do PWN kontraktu. V našem

návrhu by se nabídky na půjčky ukládaly jen na webový server. Nevýhodou výše popsaného návrhu je, že by se nabídky musely zobrazovat na centralizovaném webovém serveru, a tím se zmenší decentralizovanost aplikace.

5.7 Návrhy pro vylepšení kódu

V této podkapitole poukážeme na drobné a nezávažné problémy v kódu, chybné nebo chybějící komentáře a uvedeme návrhy, jak to napravit.

- Řádek 134 v *PWNDeed.sol* obsahuje špatný komentář. Funkce *revoke* nespálí token, pouze ho uvede do stavu 0.
- Na řádku 240 v *PWNDeed.sol* je špatná chybová hláška ve funkci *repayLoan*. Tato chybová hláška se může vypisovat i v momentě, kdy je přijatá nabídka, ale token je již expirovaný. Poté je text chybové hlášky zavádějící.
- Nekontrolují se návratové hodnoty, ale to jsme již řešili dříve v této kapitole.
- Ve smart kontraktech není sjednocené formátování, doporučujeme pro formátování používat nástroj *black*.

Závěr

V diplomové práci jsem provedl bezpečnostní audit aplikace PWN, která je napsaná v jazyce Solidity a je určená k decentralizovanému propojovaní věřitelů a dlužníků. Součástí cíle bylo provést testovací nasazení aplikace na simulovaný blockchain, kde jsem otestoval její základní funkčnost. Poté jsem měl napsal unit testy, které by podle zadání měly mít pokrytí alespoň 90% pro testované funkce. Dalším cílem bylo spustit nástroj pro statickou analýzu kódu aplikace a výstup této analýzy okomentovat. Následně jsem měl podle zadání provést fuzzy testování pro parametry funkcí a pro pořadí spuštění funkcí a na závěr udělat manuální code review.

Všechny body zadání jsem splnil nebo v případě fuzzy testů jsem argumentoval, proč je psaní fuzzy testů pro tento typ aplikací nevhodné a proč jsem je neudělal.

Práce v první části popsala koncepty stojící za protokolem kryptoměny Ethereum. Vysvětluji, jak je možné dosáhnout decentralizovaného konsenzu pomocí mechanizmů proof of work a proof a stake a diskutuji výhody a nevýhody těchto dvou přístupů. Názorně ukazuji, že Ethereum používá podobné mechanizmy jako Bitcoin, ale rozšiřuje použití blockchainu na ukládání decentralizovaných aplikací, které je možné pomocí transakcí spouštět.

Pro nasazení na testovací blockchain jsem použil nástroj Brownie, který blockchain simuluje pomocí dalšího nástroje Ganache. Unit testy jsem pro funkce PWN napsal s pokrytím 100% a během testů jsem narazil na neočekávané chování, které je způsobeno odkloněním od specifikace ve white paperu. Spuštěním nástroje Slither jsem dostal seznam potencionálních zranitelností, které jsem v práci všechny rozebral, ale žádnou jsem nevyhodnotil jako reálnou bezpečnostní hrozbu. Během manuálního code review jsem se zabýval tím, že aplikace PWN používá časovou známku bloku pro spočítání okamžiku, kdy je půjčka expirovaná a dlužník nestihl splatit dluh. Popsal jsem případ, kdy by věřitel tohoto mohl zneužít a zkrátit datum splatnosti až o 15 minut. Dále jsem poukázal na dva případy, kdy se implementace PWN aplikace odklonila od specifikace ve white paperu. Jde o případ, kdy white paper tvrdí, že

aplikace nastaví datum splatnosti v momentě vytvoření půjčky, ale v implementaci se tak stane až ve chvíli, kdy je půjčka akceptovaná. Druhé nedodržení specifikace je, že není implementovaná podpora pro půjčování tokenů podle standardu ERC1155, což by podle white paperu mělo být možné. Tato odklonění považuji za velkou chybu, protože snižuje důvěryhodnost aplikace. Na konci manuálního code review kritizuji celkový návrh aplikace, který požaduje pro půjčky velké množství transakcí do blockchainu, které jsou drahé. Návrh také dovoluje zkoušet přijímat návrhy na půjčky, i když neprojdou z důvodu, že věřitel nemá dostatečný zůstatek. Dlužník však stejně za neúspěšnou transakci musí zaplatit. Představil jsem návrh aplikace, který by velkou část komunikace přenesl mimo blockchain, snížil by počet transakcí pouze na 2 a řeší i problém nedostatečného zůstatku věřitele.

Aplikaci PWN považuji za velmi dobře naprogramovanou, v kódu není žádná reálně zneužitelná zranitelnost. Ovšem v návrh aplikace vidím několik slabin, které při vysoké ceně transakcí ohrožují použitelnost celé aplikace. Za velký problém považuji nedodržení specifikace ve white paperu, který by měl upraven podle implementace aplikace.

V budoucnu vidím možnost v pokračování práce a použít informace uvedené v této práci pro analýzu kódu jiných decentralizovaných aplikací.

Literatura

[1] Takenobu T: *Ethereum EVM illustrated [online]*. 2018. Dostupné z: https://takenobu-hs.github.io/downloads/ethereum_evm_illustrated.pdf

[2] Ethereum Community: *Ethereum energy consumption [online]*. 2020. Dostupné z: https://ethereum.org/cs/energy-consumption/

[3] Ethereum Community: *PROOF-OF-STAKE (POS) [online]*. 2020. Dostupné z: https://ethereum.org/en/developers/docs/consensus-mechanisms/pos/

[4] Je, J.: *PWN Finance P2P loans backed by arbitrary collateral [online]*. 2021. Dostupné z: https://pwn.finance/PWN-Whitepaper.pdf

[5] Antonopoulos, A. M.: *Mastering bitcoin: programming the open blockchain*. O'Reilly, druhé vydání, ISBN 978-1-491-95438-6.

[6] Nakamoto, S.: *Bitcoin: A Peer-to-Peer Electronic Cash System [online]*. 2008. Dostupné z: https://bitcoin.org/bitcoin.pdf

[7] Bullion and Bandits: The Improbable Rise and Fall of E-Gold. *Wired [online]*, září 2009, [cit. 2022-16-01]. Dostupné z: https://www.wired.com/2009/06/e-gold/

[8] of Justice, D.: *Digital Currency Business E-Gold Pleads Guilty to Money Laundering and Illegal Money Transmitting Charges [online]*. 2008. Dostupné z: https://www.justice.gov/archive/opa/pr/2008/July/08-crm-635.html

[9] B-Money. *Investopedia [online]*, Říjen 2021, [cit. 2022-20-01]. Dostupné z: https://www.investopedia.com/terms/b/bmoney.asp

[10] Dai, W.: *B-money [online]*. 1998. Dostupné z: http://www.weidai.com/bmoney.txt

[11] Back, A.: *Hashcash - A Denial of Service Counter-Measure [online]*. 2002. Dostupné z: http://www.hashcash.org/papers/hashcash.pdf

[12] Wuille, P.: *Hierarchical Deterministic Wallets [online]*. 2012. Dostupné z: https://github.com/bitcoin/bips/blob/master/bip-0032.mediawiki

[13] Mansi Bosamia, D. P.: *Current Trends and Future Implementation Possibilities of the Merkel Tree [online]*. 2018. Dostupné z: https://www.researchgate.net/publication/327601654_Current_Trends_and_Future_Implementation_Possibilities_of_the_Merkel_Tree

[14] Uknown: *Bitcoin Block Reward Halving Countdown [online]*. 2022. Dostupné z: https://www.bitcoinblockhalf.com/

[15] Pieter Wuille, T. R., Jonas Nick: *Schnorr Signatures for secp256k1 [online]*. 2020. Dostupné z: https://github.com/bitcoin/bips/blob/master/bip-0340.mediawiki

[16] Abdul Ghaffar Khan, M. U. R., Sana Basharat: *Analysis of asymmetric cryptography in information security based on computational study to ensure confidentiality during information exchange [online]*. 2018. Dostupné z: https://www.researchgate.net/publication/328630416_Analysis_of_asymmetric_cryptography_in_information_security_based_on_computational_study_to_ensure_confidentiality_during_information_exchange

[17] S.R. Subramanya, B. Y.: *Digital signatures [online]*. 2006. Dostupné z: https://www.researchgate.net/publication/3227862_Digital_signatures

[18] Darrel Hankerson, S. V., Alfred Menezes: *Guide to Elliptic Curve Cryptography*. Springer, první vydání, ISBN 0-387-95273-X.

[19] Milne, J.: *Elliptic Curves*. BookSurge Publishers, 2006, ISBN 1-4196-5257-5.

[20] Tomáš Kalvoda, S., Ivo Petr: *Matematika pro kryptologii [online]*. 2019. Dostupné z: https://courses.fit.cvut.cz/MI-MKY/lectures/files/mi-mky-poznamky-v17.pdf

[21] Barker, E.: *Recommendation for Key Management [online]*. 2020. Dostupné z: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57pt1r5.pdf

[22] Andreas M. Antonopoulos, G. W.: *Mastering Ethereum*. O'Reilly, 2019, ISBN 978-1-491-97194-9.

[23] Ethereum Community: *INTRO TO ETHEREUM [online]*. 2020. Dostupné z: https://ethereum.org/en/developers/docs/intro-to-ethereum/

[24] WOOD, G.: *ETHEREUM: A SECURE DECENTRALISED GENERALISED TRANSACTION LEDGER [online]*. 2016. Dostupné z: http://gavwood.com/paper.pdf

[25] Ethereum Community: *ETHEREUM ACCOUNTS [online]*. 2020. Dostupné z: https://ethereum.org/en/developers/docs/accounts/

[26] Ethereum Community: *TRANSACTIONS [online]*. 2020. Dostupné z: https://ethereum.org/en/developers/docs/transactions

[27] Ethereum Community: *BLOCKS [online]*. 2020. Dostupné z: https://ethereum.org/en/developers/docs/blocks/

[28] Ethereum Community: *Mining [online]*. 2020. Dostupné z: https://docs.ethhub.io/using-ethereum/mining/

[29] Buterin, V.: *Ethereum: A Next-Generation Smart Contract and Decentralized Application Platform [online]*. 2014. Dostupné z: https://ethereum.org/669c9e2e2027310b6b3cdce6e1c52962/Ethereum_White_Paper_-_Buterin_2014.pdf

[30] Ethereum Community: *Modified Merkle Patricia Trie Specification (also Merkle Patricia Tree) [online]*. 2020. Dostupné z: https://eth.wiki/en/fundamentals/patricia-tree

[31] Ethereum Community: *GAS AND FEES [online]*. 2020. Dostupné z: https://ethereum.org/en/developers/docs/gas/

[32] Ethereum Community: *ETHEREUM VIRTUAL MACHINE [online]*. 2020. Dostupné z: https://ethereum.org/en/developers/docs/evm/

[33] Ethereum Community: *MINING [online]*. 2020. Dostupné z: https://ethereum.org/en/developers/docs/consensus-mechanisms/pow/mining/

[34] Ethereum Community: *Contract ABI Specification [online]*. 2020. Dostupné z: https://docs.soliditylang.org/en/v0.8.13/abi-spec.html

[35] Joseph C: *The Beacon Chain Ethereum 2.0 explainer you need to read first [online]*. 2020. Dostupné z: https://ethos.dev/beacon-chain/

[36] Ethereum Community: *The Merge [online]*. 2020. Dostupné z: https://ethereum.org/en/upgrades/merge/

[37] Ethereum Community: *Shard chains [online]*. 2020. Dostupné z: `https://ethereum.org/en/upgrades/shard-chains/`

[38] Ethereum Community: *STATE CHANNELS [online]*. 2020. Dostupné z: `https://ethereum.org/en/developers/docs/scaling/state-channels/`

Příloha **A**

Seznam použitých zkratek

DoS Denial of Service

UTXO Unspent transaction outputs

UI User interface

POW Poof of work

SPV Simplified payment verification

EC Elliptic Curve

ECC Elliptic Curve Cryptography

GF Galoisovo těleso

ECDSA Digitální podpis na eliptických křivkách

ETH Ether

EVM Ethereum Virtual Machine

MPT Merkle-Patricia tree

UB Uncles Blocks

DoS Denial of Service

POS Proof of stake

ETH2 Ethereum 2.0

ABI Aplication binary interface

DeFI Decentralized Finance

NFT Non-fungible token

Příloha B

Obsah přiloženého CD

- `readme.md` ..... stručný popis obsahu datového nosiče
- `src` ..... zdrojové kódy
  - `build` ..... ABI smart kontraktů po kompilaci
  - `contracts` ..... zdrojové kódy ke smart kontraktům psané v Solidity
  - `reports` ..... pokrytí funkcí testy
  - `scripts` ..... python skripty pro nasazení pomocí Brownie
  - `slither` ..... zdrojové kódy a výstup z nástroje Slither
  - `tests` ..... zdrojové kódy testů
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF